DRAFT

Doing Computational Phonology

July 3, 2019

DRAFT

© Jeffrey Heinz

Contents

DRAFT

DRAFT

 © *Jeffrey Heinz*

DRAFT

Part I

Foundations

DRAFT

DRAFT

Chapter 1

Intensional and Extensional Descriptions of Phonological Generalizations

Jeffrey Heinz

1.1 Generative Phonology

Within languages, the pronunciation of a morpheme often differs depending on the word in which it occurs. Examples like English *go/went* may indicate that these different pronunciations have almost nothing in common, it is much more typical that the pronunciations of the same morpheme in different words are in fact similar, as with common English plural *cat[s]/dog[z]*. The main empirical conclusion linguists have drawn is that the variation in the pronunciation of morphemes is *systematic*. It is no accident that the plural form of *tip* uses [s] just like *cat[s]* and that the plural form of *dud* is [z] just like *dog[z]*. Explaining this systematic variation is thus an important goal of linguistic theory.

The central hypothesis of Generative Phonology (GP) holds is presented below.

† The observed systematic variation in the pronunciation of morphemes is best explained if people hold a single mental representation of the pronunciation of each morpheme (the underlying representation, UR) which is *lawfully transformed* into its pronounced variants (the surface

representation, SR).

This book assumes this hypothesis is correct, and does not review any arguments for it.[1] Readers interested in arguments for this position are directed to Odden (2014, chapter 4) and Kenstowicz and Kisseberth (1979, chapter 6).

If this hypothesis is correct, then there are three questions every theory of generative phonology must address:

1. What is the nature of the underlying representations?
2. What is the nature of the surface representations?
3. What is the nature of the transformations between these representations?

These questions are certainly not exhaustive but they are centrally important. For instance, another important question "How different can the underlying representations be from the surface representations?" (the question of abstraction) has been raised and studied (Kenstowicz and Kisseberth, 1977).

This book provides a general framework which addresses these questions from a *computational* perspective. The computational perspective addresses both the nature of the representations and the nature of the transformations. It is flexible in the sense that different representational schemes can be studied and compared. This is accomplished through *model-theoretic* representations of words and phrases. It is also flexible in the sense that different types of computational power can be studied and compared. This is accomplished by studying what can be accomplished with logical expressions of different types. As will be explained, model theory and logic provide a mathematical foundation for theory construction, theory comparison, and even descriptive linguistics.

The study of phonology from the computational perspective allows one to construct theories of phonology which provide answers to the above questions. Representational choices and choices of logical power essentially determine the theory and its empirical predictions. Theories of phonology developed

[1] The words *transformed* and *transformation* are used here in their original meaning simply to signify that the URs become SRs, and that the SR derived from some UR may not be identical to this UR. If a UR is related to a SR via the transformative component of a phonological grammar, it is also often said the UR is mapped to the SR. These words are deliberately neutral with respect to the specific type of grammar being employed.

DRAFT

under this framework are examples of Computational Generative Phonology (CGP).

To begin motivating CGP, I would like to give some examples of how phonological theories aim to answer these questions. It is not possible to comprehensively survey here the range of answers that have been offered. Therefore, I only highlight some answers (and only in very broad strokes).

Rule-based theories, as exemplified by Chomsky and Halle (1968a), for example, have argued that the abstract underlying representations are subject to language-specific morpheme structure constraints (MSCs). The transformation from underlying forms to surface forms are due to language-specific rules, which are applied in a language-specific order. Constraints on surface representations were, generally speaking, not part of the ontology of these theories, and therefore were not posited to have any psychological reality. Such generalizations—the phonotactic generalizations—were derivable from the interaction of the MSCs and the rules.

On the other hand, in classic Optimality Theory (Prince and Smolensky, 1993, 2004), there are no constraints on underlying representations (richness of the base), but there are psychologically real, universal constraints on surface forms (markedness constraints). The transformation from underlying forms to surface forms is formulated as an *optimization* over these markedness constraints, in addition to constraints which penalize differences between surface and underlying forms (faithfulness constraints). While both the markedness and faithfulness constraints are universal, their relative importance is language-specific. So in every language the surface pronunciation of an underlying representation is predicted to be the optimal form (the one that violates the most important constraints the least). Of course what is optimal varies across languages because the relative importance of the constraints may vary across languages.

These two theories are radically different in what they take to be psychologically real. The ontologies of the theories are very different. Perhaps this is most clear with respect to the concept of phonemes (Dresher, 2011). Phonemes exist as a consequence of the ontology of rule-based theories, but they do not as a consequence of the ontology of OT. This is simply because phonemes are a kind of MSC; underlying representations of morphemes must be constructed out of them, and nothing else. In OT, there are no MSCs and hence there are no phonemes. The principle of Lexicon Optimization guarantees that the URs of *pit* and *spit* are [pʰɪt] and [spɪt], respectively (Kager, 1999). The underlying, mental representation of the voiceless labial stops in

DRAFT

 © Jeffrey Heinz

both words are not the same. Consequently, the complementary distribution of speech sounds are explained in a very different manner in the two theories, and these theories promote different views of the notion of *contrast*. Despite these differences however, there is an important point of agreement: In both theories, complementary distribution of speech sounds in surface forms is the outcome of a transformation of underlying forms to surface forms.

This is the point I wish to emphasize: neither theory abandons the fundamental insight stated in 1.1.[2] The theories offer radical different answers to the questions in 1.1, but *they agree on the questions being asked*.

In the remainder of this chapter, I motivate a computational approach to phonology. I first make an important distinction between extensional and intensional descriptions of linguistic generalizations and argue that the former is important for understanding the latter. I then argue that neither rule-based or constraint-based formalisms as practiced provide adequate intensional descriptions of phonological generalizations.

This is then contrasted with automata and logical descriptions of language. The chapter concludes that logical descriptions of linguistic generalizations are preferable to automata-theoretic descriptions for several reasons. This is not to say automata are not useful (they are!) but that logic offers more in the short term to linguists interested in writing and analyzing grammars. So when we consider the ways in which we spend our time, logic is a good place to start.

DRAFT

1.2 Extensional and Intensional Descriptions

McCarthy (2008, pp. 33–34) emphasizes the importance of descriptive generalizations in preparing analyses. "Good descriptive generalizations," he writes "are accurate characterizations of the systematic patterns that can be observed in the data." They are, as he explains, "the essential intermediate step between data and analysis." This is because descriptive generalizations go beyond the data; they make predictions about things not yet observed.

Descriptive generalizations are important for computational phonology too. They are typically stated in prose. For example, consider the phonological generalizations below.

[2]It is true that periodically some work is published in that direction, for example the work on output-to-output correspondence (Benua, 1997, and others).

 © *Jeffrey Heinz*

1. Word final vowels are prohibited.

2. Consonant clusters are prohibited word-finally.

These generalizations are good ones because they allow the analyst to recognize that potentially unobserved forms like *tapaka* is ill-formed but *tanak* is well-formed with respect to 1. Similarly, we recognize that 2 distinguishes between forms like *tapakt* and *tanakta*.

The generalizations above divide every word of every length cleanly into two sets: those that obey the description and those that do not. This is illustrated in the figure below. The set of words that is well-formed according

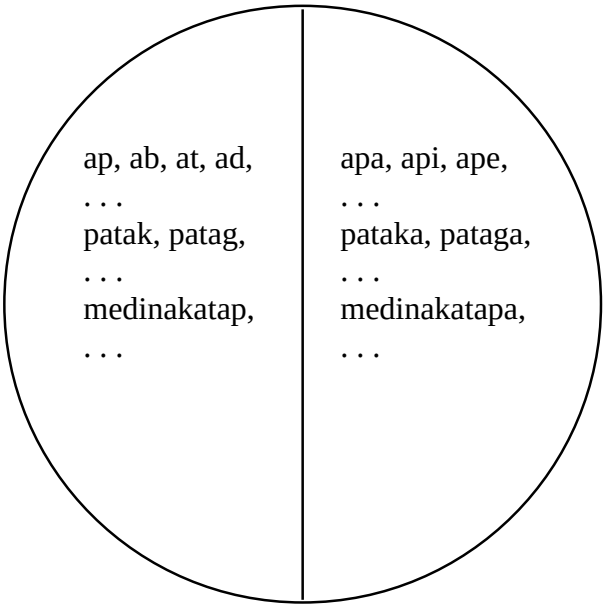

Figure 1.1: Generalizations about well-formedness partition the set of all possible forms.

to 1 is called the *extension* of 1.

Importantly, this set—the extension—is infinite in size. For instance, it is not possible to write down every word that obeys 1. If a set of words formed from a finite alphabet is infinite then there is no upper bound on the length of words. Likewise, if there is no upper bound on the length of words, then the set of words formed from a finite alphabet is also infinite. Thus whether the size of a set of words is infinite or not is intertwined with whether or

DRAFT

not there is an upper bound on the length of words. These issues are so important to get clear that they are discussed in further detail below.

Extensional descriptions contrast with *intensional descriptions* of generalizations. For now, intensional descriptions can be thought of as grammars that denote the extension. The prose in 1 and 2 are examples of intensional descriptions. Rule-based grammars and OT grammars are also examples of intensional descriptions. A good intensional description is one where the the extension can be rigorously and precisely defined from the intensional description. Generally, English prose does not make for good intensional descriptions. Further below, I will argue that in their current forms and practice, rule-based grammars and OT grammars are more like English prose than good intensional descriptions.

Let us now return to the infinitely-sized extensions. Is it reasonable for descriptive generalizations 1 to denote an infinite set of words? Yes, it is. One reason is that these generalizations make no reference to length at all. If the length of words mattered, it ought to be part of the generalization. Another way of thinking about this is that if there were a principled upper bound on the length of words, then that would be a generalization *distinct* from 1 above, and hence ought not be included within it. Finally, even if for some reason 1 ultimately denoted a finite set, there are reasons to treat its extension as infinite anyway. Savitch (1993) argues that large finite sets of strings are often best understood if they are factored into two parts: an infinite set of strings and a separate finite-length condition. They are, in his words, "essentially infinite." The basis of the argument is a demonstration that intensional descriptions of infinite sets can be smaller in size than the intensional descriptions of finite sets.

These infinite-sized extensions do not exist in the same way that your fingernails, your bed, or your brain exists. Instead they exist mathematically. Each generalization is an infinite object like a circle, a set of infinitely many points each exactly the same distance from a center. But we can never see the mathematical object in its entirety in the real world. It is a fact that circles as infinite objects do not exist. The situation with linguistic generalizations is similar. The extension is there mathematically, but we cannot write down every element of the extension in a list for the same reason all points of a circle cannot be written down in a list since there are infinitely many. But we can write down a grammar which can be understood as generating the infinite set, in the same way that a perfect circle can be generated by specifying a center point and a distance, the radius.

DRAFT

The same circle can be described in other ways as well. If we employ the Cartesian plane, we could generate a circle with an equation of the form $(x-a)^2+(y-b)^2=r^2$ where the r is the radius of the circle and (a,b) is its center. The equation is interpreted as follows: all and only points (x,y) which satisfy the equation belong to the circle. The equation is an intensional description and the set of points, the circle, is its extension.

We can also describe a circle on a plane with polar coordinates instead of Cartesian ones. Recall that polar coordinates are of the form (r,θ) where r is the radius and θ is an angle. The equation $r=2a\cos(\theta)+2b\sin(\theta)$ provides the general form of the circle with the radius given by $\sqrt{a^2+b^2}$ and the center by (a,b) (in Cartesian coordinates). The polar equation is interpreted like the Cartesian one: all and only points (r,θ) which satisfy the equation belong to the circle.

There are some interesting differences between these two coordinate systems. Each point in the Cartesian system has a unique representation, but each point in the polar system has infinitely many representations (since the same angle can be described in infinitely many ways, e.g. $0^\circ = 360^\circ = 720^\circ = \ldots$). If the center of the circle is the origin, the polar equation simplifies to $r=a$ whereas the Cartesian equation remains more complicated $x^2+y^2=r^2$. Thus, the polar equation $r=4$ and the Cartesian equation $x^2+y^2=16$ are different equations with different interpretations, but they describe the same unique circle: one of radius four centered around the origin. The two equations differ intensionally, but their extension is the same.

It seems strange to ask which of these two descriptions is the 'right' description of a circle. They are different descriptions of the same thing. Some descriptions might be more useful than others for some purposes. It also interesting to ask what properties the circles have irrespective of a particular description. For instance the length of the perimeter and the area of a circle are certainly relatable to these descriptions, but they are also in a sense independent of the particulars. The perimeter and area depend on the radius but not the center, though both the radius and the center appear in the equations. This suggests that the radius is a more fundamental structure to a circle than its center, though both certainly matter.

The analogy I wish to draw is that rule-based and OT-theoretic formalisms are like the Cartesian and polar systems. The analogy is far from perfect, but it is instructive. Both rule-based and OT analyses provide descriptions of platonic, infinitely sized objects. In many cases, but not all, the two formalisms describe the same object, insofar as the empirical evidence

DRAFT

 © Jeffrey Heinz

allows.

What is this object? The transformations from underlying forms to surface forms can be thought of as a *function*, in the mathematical sense of the word. Another word for function becoming prevalent in the phonological literature is *map* (Tesar, 2014). For example, consider the descriptive generalizations below.

1. Word final vowels delete.

2. Word final vowels delete except when preceded by a consonant cluster.

These generalizations also have infinite-sized extensions, but the extensions are better understood as functions.

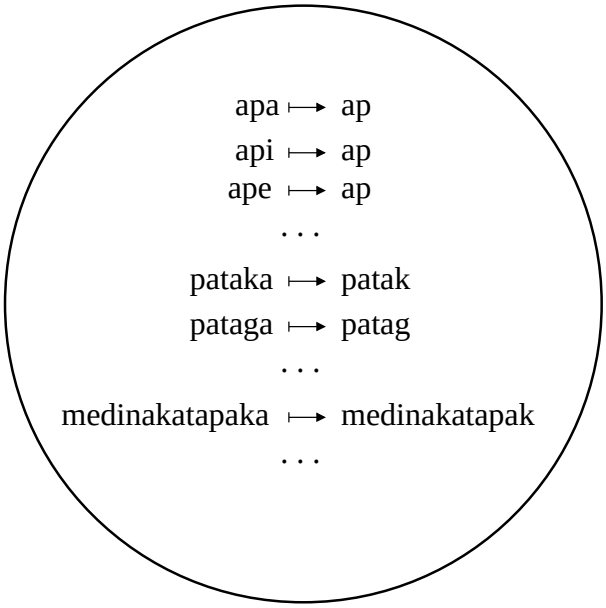

Figure 1.2: Generalizations about transformations are functions.

There are three parts to a function. One, there is its domain, which is the set of objects the function applies to. Two, there is its co-domain, which is the set of objects to which the elements of the domain are mapped. Three, there is the map itself, which says which domain elements are transformed to which co-domain elements. Thus to specify a function, one needs to provide a description of its domain, its co-domain, and a description of which

DRAFT

domain elements become which co-domain elements. Following traditional phonological terminology, I use the term *constraint* to refer to intensional descriptions of either the domain or co-domain.

This lines up nearly perfectly with the fundamental questions of phonological theory. The underlying representations correspond to the domain. The surface representations are the co-domain. And the transformation from underlying to surface forms is the map from domain elements to co-domain elements. From this perspective, describing the phonology of a language requires describing aspects of this function.

Further, in linguistic typology we are actually interested in the *class* of such functions that correspond to *possible* human phonologies. If the phonologies of languages are circles we would be interested in the universal properties of circles and the extent of their variation. Circles are pretty simple, so the answers are straightforward. All circles have a center and a radius, but their centers can be different points and their radii can have different lengths. What universal properties do phonological functions share? What kind of variation does the human animal permit across these functions?

The point is that when we develop a linguistic generalization, it is important to know what its extension is. Ultimately, the intensional (grammatical) description we provide must generate this extension. The emphasis placed here on the extensional description as an infinite object should not be taken to mean intensional descriptions do not matter. Of course they matter – theories of these intensional descriptions ought to make predictions about what is psychologically real, predictions that in principle are testable with the right kinds of psycholinguistic and neurolinguistic experimentation. They also make predictions about linguistic typology: the available intensional descriptions limit the extensions accordingly. In addition to making correct predictions, phonologists expect that intensional descriptions express the 'right' generalizations.

Extensional descriptions are an essential, intermediate step between the prose descriptive generalizations and the formal intensional descriptions (the grammatical analysis).

It is critically important that it is well-understood how the intensional descriptions relate to the extensional ones. We want to be able to answer questions like the following:

1. Given a word w and an intensional description of a constraint C, does w violate C? (We may also be interested in the number of violations

DRAFT

 © Jeffrey Heinz

and their locations in the word.)

2. Given a word w in the domain of a transformation f what words in the co-domain of f does f map w to, if any?

3. Given a word v in the co-domain of a transformation f what words in the domain of f map to v, if any?

Question 1 is often called the membership problem. Question 2 is often called the generation problem. Question 3 is often called the recognition or parsing problem. Good intensional descriptions allow answers to these questions to be computed effectively. In the next section, I argue that rule-based intensional descriptions and OT grammars are not good intensional descriptions in this narrow sense.

DRAFT

1.3 Issues with Familiar Grammars

Chomsky and Halle (1968b) present a formalization based on rewrite rules. The basic rewrite rule is of the form $\text{A} \longrightarrow \text{B} \;/\; \text{C} \_\_ \text{D}$. This notation is intended to mean that if an input string contains CAD then the output string will output CBD (so A is rewritten as B in the context C $\_\_$ D). To understand the extension of a rule, we need to know how to apply it. Originally, Chomsky and Halle (1968a, p. 344) intended for the rules to apply simultaneously to all the relevant targets in an input string. They wrote, "To apply a rule, the entire string is first scanned for segments that satisfy the environmental constraints of the rule. After all such segments have been identified in the string, the changes required by the rule are applied simultaneously." For many phonological rules, this explanation appears sufficient to denote the extension. For instance the rules corresponding to the descriptive generalizations (1) is $\text{V} \longrightarrow \varnothing \;/\; \_\_ \;\#$. Humans have no difficulty using this rule to answer the generation and parsing problems above given this intensional description. However, it is much less clear what the extension of *any* rule would be.[3]

The phonological literature after SPE addressed the question of rule application (Anderson, 1974), and other types of rule application were identified such as left-to-right or right-to-left. It was clear that the mode of application determined the extension of the rule. For example, for the input string

[3]Of course this depends in part on what A, B, C and D themselves are able to denote.

 © Jeffrey Heinz

iana and rule V $\longrightarrow$ [+nasal] / $\_\_$ [+nasal] simultaneous application yields output *iãna* but right-to-left application yields output *ĩãna*. While linguistically-chosen examples served to distinguish one mode of application from another, general solutions to the generation and recognition questions by Johnson (1972) and Kaplan and Kay (1994) were for the most part ignored by generative phonologists.

It is my contention that rule application is still not well-understood by most students of phonology, despite the careful computational analyses by Johnson (1972); Kaplan and Kay (1994) and Mohri and Sproat (1996). In informal surveys of phonologists in-training, many have difficulty of applying the rule aa $\longrightarrow$ b simultaneously to the input *aaa*. People wonder whether the right output is *ab*, *ba*, or *aa*. According to Kaplan and Kay's analysis, there are two outputs for this input when the rule aa $\longrightarrow$ b is applied simultaneously. They are *ab* and *ba*. Their analysis translates rewrite rules into finite-state automata, which are grammars whose extensions are very well defined and understood. These will be explained in a bit more detail in the next section.

Interestingly, Kaplan and Kay's analyses of rule application, which has been implemented in software programs like `xfst` (Beesley and Kartunnen, 2003) and `foma` (Hulden, 2009a,b), do not exhaust the possible natural interpretations of the rewrite rule A $\longrightarrow$ B / C $\_\_$ D . Like Johnson and Kaplan and Kay's analyses, Chandlee's (2014) analysis also uses finite-state automata to determine an extension of a rule A $\longrightarrow$ B / C $\_\_$ D , provided that CAD is a finite set of strings. Unlike Kaplan and Kay, her interpretation of the extension of the rule aa $\longrightarrow$ b maps input *aaa* to *bb*. This result is arguably what Chomsky and Halle in mind when they described simultaneous application because each *aa* sequence satisfies "the environmental constraints of the rule."

The point of the foregoing discussion is simply this: a rule A $\longrightarrow$ B / C $\_\_$ D underdetermines its extension. The extensions are a critical part of any rule-based theory and there is more than one way such rules determine extensions. This point is not news nor is it controversial. It is a well-known chapter in the history of phonological theory. Chandlee (2014) shows a good understanding of the extensions of SPE-style rules is not a closed chapter in a phonological theory based on rewrite rules. Bale and Reiss (2018) may be the first textbook on phonology that provides an adequate interpretation of the application of rewrite rules.

Optimality Theory is an improvement in some sense. Given an OT gram-

DRAFT

mar, and an input form there is a well-defined solution to the generation problem. This solution follows from the architecture of the OT grammar. The GEN component generates the set of possible candidates and the EVAL component uses the grammar of ranked constraints to select the optimal candidates.

Nonetheless in actual phonological analyses the generation problem faces two difficulties, each acknowledged in the literature. The first one is ensuring that all the possible candidates are actually considered by EVAL. The absence of an overlooked candidate can sink an analysis. The proposed optimal candidate turns out to be less harmonic than some other candidate that the analysts failed to consider. How can analysts ensure that every candidate has been considered?

The second is ensuring that all the relevant constraints are present in the analysis. The absence of a relevant constraint can also sink an analysis. (Prince, 2002, p. 276) makes this abundantly clear. He explains that if a constraint is ignored that must be dominated by some other constraint then the analysis is "dangerously incomplete." Similarly, if a constraint is omitted that may dominate some other constraint then the analysis is "too strong and may be literally false."

As a result, any phonological analysis of a language which does not incorporate the entire set of constraints is not guaranteed to be correct. This makes studying some aspect of the phonology of the language difficult. The constraints deemed irrelevant to the fragment of the phonology under investigation (and which are therefore excluded) actually need to be shown to be irrelevant for analysts to establish the validity of their analyses.

Both these problems in OT can be overcome. The solution again comes from the theory of computation, in particular from the theories of finite-state automata and so-called regular languages (defined and discussed in the next section). The primary result is that even if the constraints and GEN can be defined in these terms, the maps OT produces are not guaranteed to be definable in these terms — unless the constraints have a finite bound on the maximum number of violations they can assign (Frank and Satta, 1998). Karttunen (1998) uses this fact to provide a solution and software for the generation and recognition problems (see also (Gerdemann and Hulden, 2012)), and he assumes each constraint has some maximum number of violations. While some theoretical phonologists have argued for this position (McCarthy, 2003), most do not adopt it. Riggle (2004) provides a different solution which does not require bounding the number of violations

DRAFT

 © Jeffrey Heinz

constraints assign. His solution is guaranteed to be correct provided the map the OT grammar is in fact representable as a finite-state relation (not all of them are). Another solution is present in Albro's (2005) dissertation, which provides a comprehensive OT analysis of the phonology of Malagasy.

Each of these authors make use of finite-state automata to guarantee the correctness of their solutions. However, none of these approaches have yet to make its way into the more commonly used software for conducting OT analyses such as OTSoft (Hayes *et al.*, 2013), OT-Help (Staubs *et al.*, 2010), and OTWorkplace (Prince *et al.*, 2016). A particular weakness of this software, unlike Karttunen's, Riggle's, and Albro's is that they can only work with finite candidate sets, despite the fact that GEN is typically understood as generating an infinite candidate set. Consequently, the commonly used software amounts to nothing more than pen-and-paper approaches with lots of paper and lots of pens, and so the aforementioned issues remain (Karttunen, 2006).

McCarthy (2008, p. 76) argues the aforementioned computational approaches are only possible in "narrowly circumscribed phenomenon," which ignores Albro's detailed, thorough analysis of the whole phonology of Malagasy. He also argues the methods are only as good as the algorithm that generates the candidates. That may be true, but the alternatives are manual, heuristic methods.[4] People may differ on which is better, but I will place my bets on the algorithm which is guaranteed not to leave out candidates that GEN produces. McCarthy's dismissal of the value of computational approaches is unfortunate, but it is representative of attitudes in the field.

Regardless of the extent the which different researchers appreciate the computational treatments of phonological theories, it is noteworthy and no accident that every attempt to guarantee a solution of the recognition and generation problems (and the membership problem when constraints are involved) makes use of finite-state automata and the theory of regular languages. Even OTWorkplace employs the finite-state calculus (with regular expressions) to automatically assign candidates constraint violations. What are these devices and what makes them so good for denoting extensions of generalizations?

[4]It is true that the GEN function in the Albro's, Karttunen's, and Riggle's methods is not exactly the same as the one assumed in Correspondence Theory (McCarthy and Prince, 1995), but it is instructive to understand why.

 © Jeffrey Heinz

DRAFT

1.4 Computational Theory of Language

Automata are a cornerstone of the computational theory of language. Automata are machines that process specific types of data structures like strings or trees. They form a fundamental chapter of computer science. There are many kinds of automata. The Turing machine is just one example. Pushdown automata are another. Readers are referred to texts such as Sipser (1997) and Hopcroft *et al.* (2001) for overviews of the theory of computation.

There are also deep connections between automata and logic. In this section, I will briefly review finite-state automata for string processing. Then I will informally introduce logic as another way of providing an intensional description of phonological generalizations. Their extensions are also well-defined; and in fact in many cases there are algorithms which convert a logical description into an automaton that describes exactly the same extension.

We begin with a simple automaton, the finite-state acceptor. It is an intensional description with a well-defined extension. As a matter of fact, it is a precise finite description of a potentially infinite set of strings.

A finite-state acceptor contains a finite set of states. We give the states names so we can talk about them; for instance they are often indexed with numbers. Some states are designated 'start' states. Some states are designated 'accepting' states. (Some states can be both 'start' and 'accepting' states.) Transitions lead from one state to another; they are labeled with letters from some alphabet. That's it. So a finite-state acceptor is of finite size. What is its extension? Well the extension is defined as follows. A word w is accepted/generated/recognized by a finite-state acceptor A if there is a path along the transitions of A which begins in a start state of A, which ends in a final state of A, and which spells out w exactly.

As an example, consider Figure 1.3, which shows the finite-state acceptor for the generalization that word-final vowels are prohibited. Per convention, the start state is designated by the unanchored incoming arrow and final states are marked with a double perimeter. The word *nok* is generated by this machine since there is a path beginning in a start state and ending in a final state which spells it out. This path is shown below.

Input:		n		o		k	
States:	0	$\rightarrow$	1	$\rightarrow$	0	$\rightarrow$	1

A minute of inspection reveals that every path for every word which ends in a vowel ends in state 0, which is not an accepting state. But every path for

DRAFT

 © Jeffrey Heinz

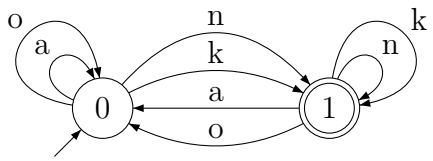

Figure 1.3: A finite state acceptor for the generalization "Word final vowels are prohibited." A simple alphabet {n,k,a,o} is assumed.

every word which does not end in a vowel ends in state 1, which is accepting. Algorithms which solve membership problem for finite-state acceptors are well understood (Hopcroft *et al.*, 2001).

Finite-state automata are not limited to acceptors. String-to-string functions can be described with automata that are called transducers. These are acceptors whose labels have been augmented with an additional coordinate. Labels are now pairs instead of a single point. Figure 1.4, which shows the finite-state acceptor for the generalization that word-final vowels delete. As before, valid paths through this machine (those that begin in start states and end in accepting states) spell out input words and the output word they map to. In the figure, the colon separates the left coordinate (input) from the right coordinate (output). The symbol λ denotes the empty string. To

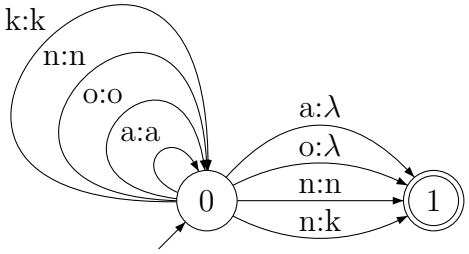

Figure 1.4: A finite state acceptor for the generalization "Word final vowels delete." A simple alphabet {n,k,a,o} is assumed.

illustrate, consider the path which shows that the output of *nako* is *nak*.

Input:		n		a		k		o	
States:	0	$\rightarrow$	0	$\rightarrow$	0	$\rightarrow$	0	$\rightarrow$	1
Output:		n		a		k		λ	

As with the membership problem and finite-state acceptors, there are algorithms which solve the generation and recognition problems for finite-state transducers.

 © Jeffrey Heinz

DRAFT

There are some interesting things to observe about the finite-state transducer. The first is that it is non-deterministic. This means for a given input, there is more than one path. For instance, the input *nok* maps to *nok*, and there are two paths that spell it out. But only one is valid: the one that reads and writes *k* and moves from state 0 to state 1.[5]

Another point is that the transducer in Figure 1.4 maps the input word *nakao* to *naka*. Thus, this machine provides the extension of the rule V $\longrightarrow \varnothing$ / $\_\_$ # applying simultaneously. In OT, if Final-C outranks Max, then the output would be *nak* with the last two vowels deleting. With rules, this could be accomplished by applying the former rule right-to-left. The finite-state transducer shown in Figure 1.5

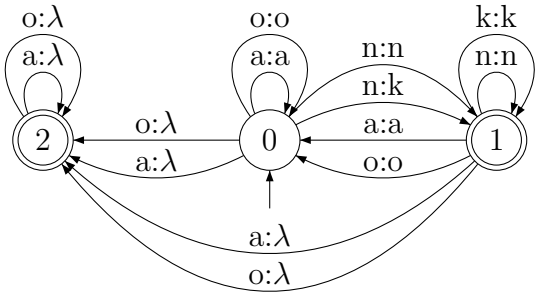

Figure 1.5: A finite state acceptor for the generalization "Strings of vowels word-finally delete." A simple alphabet {n,k,a,o} is assumed.

Transducers can also map strings to numbers. The simple one shown in Figure 1.6 counts the number of *o*s in a word. The idea here is that instead of combining the output side of valid paths with concatenation as for strings, they are combined with addition. Below is an example of the only valid path for the word *naoko* which would be mapped to 2.

Input:		n		a		o		k		o	
States:	A	→	A	→	A	→	A	→	A	→	A
Output:		0		0		1		0		1	

This is exactly the approach used by Riggle (2004) to define markedness and faithfulness constraints in OT. Again, the extension of the transducer in Fig-

[5]Non-determinism is one way optionality can be handled with finite-state transducers. If state A was also an accepting state then there would be two valid paths for the input *noko*: one would write the output *nako* and the other the output *nak*.

 © Jeffrey Heinz

DRAFT

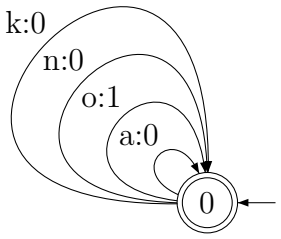

Figure 1.6: A finite state transducer which counts the number of *o*s in words. A simple alphabet {n,k,a,o} is assumed.

ure 1.5 is precisely defined and the corresponding generation and recognition problems solvable.

There are many generalizations of this kind available to transducers made possible by the study of semirings (Droste and Kuich, 2009; Goodman, 1999). Semirings are discussed in more detail in Chapter 2.

What of the recognition problem? Another important advantage of finite-state automata is that they are *invertible*. Consequently, a solution to the generation problem entails a solution to the recognition problem. Given a string *nak*, the transducer can tell you that it is the output of the any of the following inputs: *nak, naka, nako*.

Nonetheless, despite the advantage of a well-defined extensions, there are some shortcomings to using finite-state automata for phonological analyses. One is that letter of the alphabet are treated atomically. For instance, there is no sense in which the symbols [p,t,k] share any properties. It remains unclear how to incorporate phonological features and natural classes in a natural way into these machines. The most common way seems to just group the letters together that behave together as I have done int he examples above. While this is certainly sufficiently expressive, it is not satisfying. We want our intensional descriptions to somehow speak directly to the descriptive ones. In the case of "Word final vowels are prohibited" we want to be able to express this directly.

Another drawback is that as the generalizations become more complex, so do the finite-state automata. They become spaghetti-like and difficult to read. This drawback is mitigated, however, in a couple of ways. The first is that it is very well understood how to combine different finite-state automata to produce new ones. This allows the generalizations instantiated by the 'primitive' ones to persist to some degree in the complex ones. For

DRAFT

instance, it is straightforward to construct a finite-state acceptor that generates exactly the intersection of two infinite sets of strings which are generated by two acceptors. Similarly, it is straightforward to construct a finite-state transducer that generates the composition of two functions which are generated by finite-state transducers. In this way, more complex finite-state automata can be constructed from simpler parts, much in the same way more complex phonological grammars are built up from identifying generalizations that interact in some manner.

A third problem is that even simple machines are not easy to write in text. They are often pictured as diagrams, and in the same way it can be tiring to read them, it can be tiring to draw them as well. This problem is mitigated in a couple of ways. Some researchers use tables or matrix notation, others use regular expressions (Beesley and Kartunnen, 2003; Hulden, 2009b), and still others use logic.

In this book, we are going to use logic and not automata to represent linguistic generalizations. There are several reasons for this. Most importantly, like automata, the extensions of logical formula are precisely defined. Another key reason is that the representations are *flexible*. We can represent words exactly as any phonologist would want to. As this book will show, phonological features, syllable structure, autosegmental representations, phonetic information, and a host of as-of-yet unconsidered possibilities are available and directly representable with logic. Thirdly, as this book will show, the combination of logical power and representation provides a natural way to entertain distinct theories of phonology and compare them. Additionally, there is a literature showing how logical formula can be translated into automata which are equivalent in the sense that they solve the same membership, generation, and recognition problems. While this literature does not address every phonological representation proposed, the basic analytical methods which show how this can be done for strings and trees are there.

Finally, logic is not going anywhere. This is very important. If a linguist describes a generalization with logic and the representations they want, they can be guaranteed that people in will be able to read their description and understand it *hundreds of years later*.

In short, logical formula have all of the advantages, and none of the disadvantages, of automata.

DRAFT

1.5 Doing Computational Phonology

How does one do computational generative phonology? This book provides an answer.

In the first part, logical foundations and model theory are presented in the context of strings. It is explained how model theory allows one to precisely describe different representations of words and phrases. It is explained how the primitive elements in these representations would have ontological status in the theory. It is also explained how logical expressions can be used to define constraints to delimit possible representations in words and phrases and transformations to show how one representation is mapped to another. It is explained how weighted logical expressions allow ones to express a variety of linguistic generalizations, including gradient ones. These definitions and techniques are illustrated with examples drawn from phonology, as well as examples showing the terrific expressivity of the framework. The first part of this books opens a large umbrella of techniques and possibilties.

In the second part, these techniques are applied to the kinds of phonology problems one finds in standard textbooks on phonology. The focus here is descriptive in the following sense. The linguist marshalls arguments from a collection of linguistic forms before her in favor of particular linguistic generalizations. These arguments are presented and then the linguistic generalizations are formalized in terms of model-theoretic representations and logic. The chapters are short, each dealing with a relatively small and straightforward phonological problems. These examples serve as models for how analysis of other small and straightforward can be analyzed within CGP.

In the third part, the chapters address a variety of theoretical issues addressing both aspects of representation and computional power. Sebastian shows how to incorporate insights from phonetically-based phonology into CGP representationally. Hwangbo shows how representing vowel height in terms of degrees of aperture leads to straightforward analysis of vowel lowering. Strother-Garcia analyzes syllable structure and the sonority sequencing principle. Rogers and company show how the stress patterns in the world's languages can be understood as particular the combination of primitive constraints, characterizes their complexity and identifies sources of complexity. Lindell and Chandlee provide a logical characterization of Input Strictly Local functions, which Chandlee showed earlier to well-characterize an important natural class of phonological transformations. Deovletian shows that the Raimy-style linearization is computationally very complex. Once the

DRAFT

 © Jeffrey Heinz

source of the complexity is identified, he suggests way to mitigate it. Payne shows the comptuational complexity of GEN is also very complex. Vu shows how transformations can also be expressed as constraint on correspondence structures. These chapters are but a small sample of the kinds of research questions and investigations that can be addressed with the tools introduced in part one.

Computational generative phonology is not hard. We believe theories of generative phonology developed in this tradition will lead to advances in our understanding of the nature of phonological grammars and the minds which know them.

DRAFT

 © Jeffrey Heinz

Chapter 2

Representations, Models, and Constraints

Jeffrey Heinz and James Rogers

DRAFT

2.1 Logic and Constraints in Phonology

In this chapter, we show how to use logic and model-theoretic representations to define constraints on the well-formedness of those representations. The power in this kind of computational analysis comes from the framework's flexibility in both the kind of logic used and the choice of representation.

As will be explained, those choices provides a "Constraint Definition Language" (CDL) in the sense of (de Lacy, 2011). Each CDL has psychological, typological, and learnability ramifications which can be carefully studied. Conversely, the psychological, typological, and learnability considerations provide evidence for the computational nature of phonological generalizations on well-formedness.

This is not the first instance logic has been used in phonological theory. In fact, there is considerable history. A notable turning point occurred in the early 1990s with the developments of two theories: Declarative Phonology and Optimality Theory.

Declarative Phonology made explicit use of logical statements in describing the phonology of a language. For instance (Scobbie *et al.*, 1996, p. 688) expressed a general principle of theories of syllables which prohibit ambisyllabicity this way: $\forall x \neg(\text{onset}(x) \wedge \text{coda}(x))$, which in English reads "For all

segments x, it is not the case that x is both a onset and a coda."

In Optimality Theory, first-order logic was often used implicitly to define constraints. For example, the definition of the constraint MAX-IO in OT given by McCarthy and Prince (1995, p. 16) is "Every segment of the input has a correspondent in the output." On page 14, they define the correspondence relation: "Given two strings S_1 and S_2, correspondence is a relation R from the elements of S_1 to those of S_2. Elements $\alpha \in S_1$ and $\beta \in S_2$ are referred to as **correspondents** of one another when $\alpha R \beta$." As will be clear by the end of this chapter, this definition of MAX-IO is essentially a statement in First Orer Logic: For all $\alpha \in S_1$ there exists $\beta \in S_2$ such that $\alpha R \beta$.

Unlike Optimality Theory, the CDLs introduced in this chapter provide language-specific, inviolable constraints. For a representation to be well-formed it must not violate any constraint. This is a property the CDLs in this chapter have in common with Declarative Phonology. Scobbie et al. explain:

> The actual model of constraint interaction adopted is maximally simple: the declarative model. In such a model, all constraints must be satisfied. The procedural order in which constraints are checked (or equivalently, in which they apply) is not part of the grammar, but part of an implementation of the grammar (as a parser, say) which cannot affect grammaticality. (Scobbie *et al.*, 1996, p. 692)

What Scobbie et al. are emphasizing is that logical specifications of grammar specify *what is being computed* as opposed to *how it is being computed*. We agree with Scobbie *et al.* (1996) that this is an attractive property of logical languages.

While this chapter, and others in this book, assume the constraints are language-specific and inviolable, it is a mistake to conclude that this line of work only applies to grammars that make binary distinctions between well-formed and ill-formed structures. In fact, *weighted* logical languages allow one to specify what is being computed when structures are going to be assigned natural numbers (for instance in the case of counting the number of times a a structure violates a constraint) or real numbers (for instance in the case of assigning some probability to a structure) (Droste and Gastin, 2009). We review the basics and provide some examples in Chapter **??**.

DRAFT

2.2 Chapter Outline

In the remainder of this chapter, we informally introduce model-theoretic representations of strings and different logics. Most mathematical details for the models and logical languages discussed in this chapter are provided in Appendix A to Part I of this book. Some readers may benefit by consulting Appendix A in parallel with this chapter. Readers for whom this does not satisfy their appetite are referred to the textbooks on logic and model theory provided in the Further Reading section below.

We focus on strings because they are widely used and well-understood. Most importantly, they are sufficient to illustrate how different CDLs can be defined and how these CDLs have consequences for psychological models, typology, and learnability. Several chapters later in the book provide concrete examples of non-string representations motivated by phonological theory. A mathematical treatment of representations and logic is given in the appendix of part I of this book. Concepts and definitions introduced here are presented there precisely and unambiguously.

First, we introduce the canonical word model, which is known as the successor model. This is followed by an informal treatment of First-Order (FO) logic. This yields the first CDL (FO with successor) and we show how to define a constraint like *NT—voiceless obstruents are prohibited from occuring immediately after nasals—in this CDL.

Next we alter the successor model so that the representations makes use of phonological features. This yields another CDL (FO with successor and features). We comment on some notable points of comparison between the two CDLs, again using the *NT constraint.

The narrative continues by discussing one typological weakness the aforementioned CDLs: they are unable to describe long-distance constraints which are arguably part of the phonological competence of speakers of some languages. This provides some motivation for a CDL defined in terms of a more powerful logic, Monadic Second Order (MSO) logic. The CDL we call 'MSO with successor and features' and we explain how it is able to define such long-distance constraints. The key is that with MSO logic it is possible to deduce that one element in a string *precedes* another element, no matter how much later the second element occurs. The availability of the precedence relation makes it possible to define long-distance constraints.

We continue to evaluate the MSO with successor CDL from a typological perspective. We argue that there are significant classes of constraints defin-

DRAFT

 © Jeffrey Heinz

able in this CDL that are bizarre from a phonological perspective. In other words, we motivate seeking a more restrictive CDL capable of describing local and long-distance constraints in phonology.

One solution is to make the precedence relation part of the representation. This model of words is called the precedence model, which stands in contrast to the successor model. We show how the CDL "FO with precedence" is also able to describe both local and long-distance constraints of the kind found in the phonologies of the world's languages.

Finally, the chapter concludes with a high-level discussion seeking to emphasize the following points. First, there is a tradeoff between representations and logical power. Second, as mentioned, the choice of representation and the choice of logic has consequences for typology, psychological reality, memory, and learnability. Third, the representations and logics discussed in this chapter are only the tip of the iceberg. Readers undoubtedly will have asked themselves "What is possible with this representation?" and "Why don't we consider this variety of logic?" Some chapters in this book address such questions. Comprehensively answering such questions, however, is beyond the scope of this book. But it is not beyond the scope of phonological theory. If some readers of this book pose and answer such questions, then this book will have succeeded in its goals.

DRAFT

2.3 The Successor Model

This section introduces the central ideas of model-theoretic representations with a concrete example. The concrete example comes from the "successor" model, which is arguably the canonical model for strings.

Model-theoretic representations provide a uniform framework for representing all kinds of objects. Here the objects under study are strings. We need to be clear about two things: what the objects are, and what counts as a successful model-theoretic representation of a set of objects.

Strings are sequences of events. If we are talking about words, the events could be given as speech sounds from the International Phonetic Alphabet. Strings are typically defined inductively. Each event corresponds is assigned some symbol. The set of symbols in use is called the **alphabet**. Each symbol on its own is a string, and if w is a string and a is a symbol then the concatenation of w and a, written wa, is also a string. This inductive definition yields a set of objects: all logically possible sequences of symbols

 © Jeffrey Heinz

of the alphabet of finite length.

A successful model theoretic-representation of a set of objects must provide a representation for each object and must provide distinct representations for distinct objects. It may be strange to ask the question "How can we represent strings?" After all, if we are talking about the string *tent* isn't *tent* itself a representation of it? It is, but the information carried in such representations is implicit. Model-theoretic representations make the information explicit.

Model-theoretic representations for objects of finite size like strings contain two parts. The first is a finite set of elements called the **domain**. The second is a finite set of relations. The relations provide information about the domain elements. The **model signature** summarizes two parts and serves to define the nature of model in terms of the information in the representation. In this book, it is written like this: $\langle D \mid R_1, R_2, \ldots R_n \rangle$.

We first show a model-theoretic representation of a word and then we explain it. While this may seem backwards to some, it seems to work better pedagogically. It can be helpful to refer to the end-product as one goes about explaining how one got there.

Figure 2.1 shows the successor model for the word *tent* in addition to a graphical diagram of it on its right. The graphical diagram puts the domain elements in circles. Edges labeled with $\triangleleft$ indicate the binary relation called "successor." Finally, the unary relations, one for each symbol in the alphabet, are shown in typewriter face above the domain elements that belong to them. Throughout this book we will often use graphical diagrams instead of displaying the literal mathematical representation on the left. The order of the relations in the signature is fixed but it is also arbitrary.

$$
\begin{aligned}
&\mathcal{M}_{tent} \\
&= \langle D \mid \mathtt{t}, \mathtt{e}, \mathtt{n}, \mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}, \triangleleft \rangle \\
&= \Big\langle \{1,2,3,4\} \mid \{1,4\}, \{2\}, \{3\}, \\
&\qquad \varnothing, \varnothing, \ldots \varnothing, \\
&\qquad \{(1,2),(2,3),(3,4)\} \Big\rangle
\end{aligned}
$$

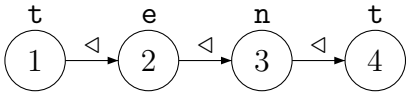

Figure 2.1: At left, the successor model of the word *tent*. At right, a graphical diagram of this model.

 © Jeffrey Heinz

DRAFT

In the case of strings, the number of domain elements matches the length of the string. So a model-theoretic representation of a word like *tent* would have a domain with four elements, one for each event in the sequence. We can represent these domain elements with the suits in a deck of cards ($\heartsuit, \diamondsuit, \clubsuit, \spadesuit$) or we could use numbers $(1, 2, 3, 4)$ as we did in Figure 2.1. We will usually use numbers because as strings get longer we can always find new numbers. However, keep in mind that the numbers are just names of elements in the model in the same way the suits would have been. They get their meaning from the relationships they stand in, not from anything inherent in the numbers themselves.

In the successor word model, there is a unary relation $\mathtt{a}$ for each symbol a in the alphabet. We use the typewriter font to distinguish the relations from the symbols. It is customary to denote the alphabet with Σ. We write $(\mathtt{a})_{a\in\Sigma}$ to mean this finite set of relations. If a domain element belongs to the unary relation $\mathtt{a}$ then it means this element has the property of being a. So for the word *tent*, two elements will belong to $\mathtt{t}$, a different element will belong to $\mathtt{e}$ and the remaining element will belong to $\mathtt{n}$. For every other symbol a in the alphabet the relation $\mathtt{a}$ will be empty. When we write $x \in \mathtt{a}$ and/or $\mathtt{a}(x)$ we mean that domain element x belongs to the unary relation $\mathtt{a}$.

There is also a single binary relation called "successor". A domain element x stands in the successor relation to y if the event y corresponds to comes in fact immediately after the event x corresponds to. In this book, we use the symbol $\lhd$ to indicate the successor relation. For the word *tent*, if $2 \in R_e$ and $3 \in R_n$ then $(2, 3)$ would be in the successor relation. We will write $(2, 3) \in \lhd$, $\lhd(2, 3)$, and/or $2 \lhd 3$ to mean that domain elements 2 and 3 stand in the successor relation.

The model signature for the successor model is thus $\langle D \mid (\mathtt{a})_{a\in\Sigma}, \lhd \rangle$. The successor model is not the only way to represent words. From a phonological perspective, it is arguably a strange model. We will consider more phonologically natural models of words below.

It is easy to see that there is a general method for constructing a unique model for each logically possible string. Given a string of w of length n we can always construct the successor model as follows. Since w is a sequence of n symbols, we let $w = a_1 a_2 \dots a_n$. Then set the domain $D = \{1, 2, \dots n\}$. For each symbol $a \in \Sigma$ and i between 1 and n inclusive, $i \in \mathtt{a}$ if and only if $a_i = a$. And finally, for each i between 1 and $n - 1$ inclusive, let the only elements of the successor relation be $(i, i + 1)$. This is summarized in Table 2.8. This construction guarantees the model's soundness: each string has a model and

 © Jeffrey Heinz

DRAFT

D	$\stackrel{\text{def}}{=}$	$\{1, 2, \ldots n\}$
a	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i = a\}$ for each unary relation a
$\lhd$	$\stackrel{\text{def}}{=}$	$\{(i, i+1) \subseteq D \times D\}$

Table 2.1: Creating a successor model for any word $w = a_1 a_2 \ldots a_n$.

distinct strings will have distinct models. It is also important to recognize that removing any one of the unary or binary relations will result in a model which does not guarantee that models of distinct strings are distinct.

Model-theoretic representations provide an ontology and a vocabulary for talking about objects. They provide a primitive set of facts from which we can reason. For instance in the word *rent*, we know that the t occurs sometime after the r. However this fact is not immediately available from the successor model. It can be deduced, but that deduction requires some computation. Measuring the cost of such computations is but one facet of what model theory accomplishes. On the other hand, the successor model makes immediately available the information that t occurs immediately after the n. As will hopefully be clear by the end of this chapter, this distinction can shed light on differences between local and long-distance constraints in phonology.

From a psychological perspective, the primitive set of facts can be thought of as the primitive psychological units. In its strongest form, the model-theoretic representation of words as embodied in its signature makes a concrete claim about the psychological reality of the ways words are represented.

2.4 First Order Logic

Now that the models provide representations, what do we do with them? Logic provides a language for talking about these representations. First Order logic is a well-understood logical language which we introduce informally here. For those already familiar with FO logic, you will see take advantage of things like prenex normal form without discussion.

In addition to the Boolean connectives such as conjunction, disjunction, implication, and negation, FO logic also includes existential and universal quantification over variables that range over domain elements. These variables are called **first order variables**. Apart from these "logical connec-

DRAFT

tives" and quantified variables, the basic vocabulary of FO logic comes from the *relations in the model signature*. Thus each model-theoretic representation supplies the ingredients for the logical language. Table 2.2 summarizes the vocabulary of FO logic with an arbitrary model $\langle D \mid R_1, R_2, \ldots R_n \rangle$. Model vocabulary are also called **atomic formulas** because they are the

Boolean Connectives	
$\wedge$	conjunction
$\vee$	disjunction
$\neg$	negation
$\rightarrow$	implication
$\leftrightarrow$	biconditional
Syntactic Elements	
(	left parentheses
)	right parentheses
,	comma for separating variables
Variables, Quantifiers, and Equality	
x, y, z	variables which range over elements of the domain
$\exists$	existential quantifier
$\forall$	universal quantifier
$=$	equality between variables
Model Vocabulary	
$\mathtt{R}(x)$	for each unary relation R in $\{R_1, R_2, \ldots R_n\}$
$\mathtt{R}(x, y)$	for each binary relation R in $\{R_1, R_2, \ldots R_n\}$
$x\mathtt{R}y$	for each binary relation R in $\{R_1, R_2, \ldots R_n\}$
...	
$\mathtt{R}(x_1, x_2 \ldots x_m)$	for each m-ary relation R in $\{R_1, R_2, \ldots R_n\}$

Table 2.2: Symbols and their meaning in FO logic. Certain sequences of these symbols are valid FO sentences and formulas. Note we write binary relations in one of two ways.

primitive terms from which larger logical expressions are built. As will be explained they play a special role in the ontology of model-theoretic linguistic theories.

DRAFT

Since the appendix defines FO logic formally, here we define valid sentences and formulas of FO logic ostensively. Below we give examples of three types of expressions: sentences of FO logic, formulas of FO logic, and syntactically ill-formed expressions.

Example 1 (Sentences of FO logic.)**.** Sentences of FO logic are complete sentences that can be interpreted with respect to a model. Below are five sentences of FO logic with English translations below.

1. Sentences of FO logic.

   (a) $\exists x, y, z\ (\neg(x = y) \wedge \neg(x = z) \wedge \neg(y = z))$
   (b) $\exists x, y\ (\mathtt{n}(x) \wedge \mathtt{t}(y) \wedge x \lhd y)$
   (c) $\neg\exists x, y\ (\mathtt{n}(x) \wedge \mathtt{t}(y) \wedge x \lhd y)$
   (d) $\forall x, y\ (\ \neg(\mathtt{n}(x) \wedge \mathtt{t}(y) \wedge x \lhd y))$
   (e) $\forall x \exists y\ (\mathtt{n}(x) \rightarrow (\mathtt{t}(y) \wedge x \lhd y))$

2. English translation (in terms of the models).

   (a) There are three distinct domain elements.
   (b) There are two domain elements in the successor relation; the former has the property of being n; the latter has the property of being t.
   (c) It is not the case that there exists two domain elements in the successor relation of which the former has the property of being n and the latter has the property of being t.
   (d) For every pair of domain elements that stand in the successor relation, it is not the case that the former has the property of being n and the latter has the property of being t.
   (e) For all domain element which have the property of being n, it is succeeded by a domain element which has the property of being t.

3. English translation (in terms of the strings the models represent).

   (a) There are at least three symbols.
   (b) There is a substring nt.
   (c) There is no substring nt.
   (d) There is no substring nt.
   (e) If there is n then there is a t immediately following it.

DRAFT

 © Jeffrey Heinz

Sentences of FO logic are **interpreted** with respect to models. Models for which the sentence is true are said to **satisfy** the sentence. If a model $\mathcal{M}$ of string w satisfies a sentence ϕ we write $\mathcal{M}_w \models \phi$. Consequently, every FO sentence ϕ divides the objects being modeled into two classes: those that satisfy ϕ and those that do not. In this way, logical sentences define **constraints**. The strings whose models satisfy the sentence do not violate the constraint; strings whose models do not satisfy the constraint do violate it.

Table 2.3 provides examples of strings whose models satisfy the formulas in Example 1 and examples of strings whose models do not. An important

ϕ	$\mathcal{M}_w \models \phi$	$\mathcal{M}_w \not\models \phi$
(a)	*too, tent, ttt*	*to, a*
(b)	*tent, rent, ntnt*	*ten, to, phobia*
(c)	*ten, to, phobia*	*tent, rent, ntnt*
(d)	*ten, to, phobia*	*tent, rent, ntnt*
(e)	*rent, antler*	*ten, nantucket*

Table 2.3: Some strings whose models satisfy the formulas in Example 1 and some whose models do not.

feature of FO logic is that there are algorithmic solutions to the problem of deciding whether a given model satisfies a given sentence. This algorithm works because the syntactic rules that build up larger sentences from smaller ones have clear semantic interpretations with respect to the model under consideration. In short, it is an unambiguous and compositional system. For instance, $\mathcal{M} \models \phi \wedge \psi$ if and only if $\mathcal{M} \models \phi$ and $\mathcal{M} \models \psi$. The interpretation of quantifiers is discussed after introducing formulas below.

Example 2 (Formulas of FO logic.)**.** Formulas of FO logic are incomplete sentences in the sense that they contain variables that are not **bound**. A variable is bound only if it is has been introduced with a quantifier and is within that quantifier's scope. Variables that are not bound are called **free**. The formulas below are only interpretable with respect to a model $\mathcal{M}$ if the free variables are assigned some interpretation as an elements of the domain of $\mathcal{M}$.

1. Formulas of FO logic.

 © Jeffrey Heinz

DRAFT

(a) $\mathtt{n}(x) \vee \mathtt{m}(x) \vee \mathtt{ŋ}(x)$
(b) $\exists y\ (\mathtt{n}(x) \wedge \mathtt{t}(y) \wedge x \lhd y)$
(c) $\neg\exists y\ (x \lhd y)$
(d) $\neg\exists y\ (y \lhd x)$
(e) $\neg(x = y) \wedge \neg(x = z) \wedge \neg(y = z)$
(f) $x \lhd y \wedge y \lhd z$

2. English translation.

(a) x has the property of being n, m, or $ŋ$.
(b) x has the property of being n and coming immediately before an element which has the property of being t.
(c) There is no element which succeeds x.
(d) There is no element which x succeeds.
(e) x, y and z are distinct.
(f) x is succeeded succeeded by y which is succeeded by z.

The difference between formulas and sentences is that sentences admit no free variables. Because these formulas can only be interpreted in terms of one or more un-instantiated variables, formulas are often used to define **predicates**. Predicates are essentially abbreviations for formulas with the unbound variables serving as parameters. Below we repeat the formulas from above, but use them to define new predicates. We write predicates in sans serif font.

$$\begin{aligned}
\mathsf{nasal}(x) &\stackrel{\text{def}}{=} \mathtt{n}(x) \vee \mathtt{m}(x) \vee \mathtt{ŋ}(x) \\
\mathsf{nt}(x) &\stackrel{\text{def}}{=} \exists y\ (\mathtt{n}(x) \wedge \mathtt{t}(y) \wedge x \lhd y) \\
\mathsf{last}(x) &\stackrel{\text{def}}{=} \neg\exists y\ (x \lhd y) \\
\mathsf{first}(x) &\stackrel{\text{def}}{=} \neg\exists y\ (y \lhd x) \\
\mathsf{distinct3}(x, y, z) &\stackrel{\text{def}}{=} \neg(x = y) \wedge \neg(x = z) \wedge \neg(y = z) \\
\mathsf{string3}(x, y, z) &\stackrel{\text{def}}{=} x \lhd y \wedge y \lhd z
\end{aligned}$$

These predicates can then be used to define new sentences. For example, the sentence $\forall x(\neg\mathsf{nt}(x))$ is equivalent to (1d) in Example 1 above. In the same way that programmers write functions which encapsulate snippets of often-used programming code, predicates generally help writing and reading complex logical sentences.

 © Jeffrey Heinz

DRAFT

Since sentences have no free variables, they must begin with quantifiers. Determining whether a model satisfies a sentence is compositional. It also depends on the **assignment** of variables to elements in the domain. For instance, to determine whether $\mathcal{M}$ satisfies $\phi = \exists x(\psi(x))$, we must find an element of the domain of $\mathcal{M}$, which if assigned to x, means that ϕ evaluates to true. If no such element exists, then $\mathcal{M}$ does not satisfy ϕ. Similarly, $\mathcal{M}$ satisfies $\phi = \forall x(\psi(x))$ if and only if every element of the domain $\mathcal{M}$, when assigned to x, results in ϕ evaluating to true.

Finally we give some examples of syntactically ill-formed sequences. The following expressions are junk; they are not interpretable at all.

Example 3 (Syntactically ill-formed sequences).

1. Syntactically ill-formed sequences.
   (a) $x\exists\ )x($
   (b) $\forall\exists\ (\mathtt{n} \vee \mathtt{t})$
   (c) $\neg\exists(\mathtt{n} \lhd \mathtt{t})$
2. Comments.
   (a) Quantifiers always introduce variables to their left and parentheses are used normally.
   (b) No quantifier can be introduced without a variable and n-ary relations from the model vocabulary must always include n variables.
   (c) Many beginning students make this sort of error when trying to express a logical sentence which forbids *nt* sequences. This expression breaks the same rules as the one before it.

We conclude this section by providing an example of a logical sentence defining a constraint which bans voiceless obstruents after nasals. This is constraint in the literature is often abbreviated *NT. Since the model signature does not include relations for concepts like nasals and voiceless consonants, we first define predicates for these notions. We assume the alphabet is limited to the following IPA symbols: *a,b,d,e,g,i,k,l,m,n,o,p,r,s,t,u,z*.

Example 4 (The constraint *NT defined under the FO with successor

 © Jeffrey Heinz

DRAFT

model.).

$$\mathsf{nasal}(x) \stackrel{\text{def}}{=} \mathtt{n}(x) \vee \mathtt{m}(x) \tag{2.1}$$

$$\mathsf{voiceless}(x) \stackrel{\text{def}}{=} \mathtt{p}(x) \vee \mathtt{t}(x) \vee \mathtt{k}(x) \vee \mathtt{s}(x) \tag{2.2}$$

$$^*\mathsf{NT} \stackrel{\text{def}}{=} \neg\exists x, y(x \lhd y \wedge \mathsf{nasal}(x) \wedge \mathsf{voiceless}(y)) \tag{2.3}$$

It is easy to see that models of words like *tent* and *lampoon* do not satisfy $^*\mathsf{NT}$ but models of words like *ten* and *moon* do. For example, in the model of *tent*, the expression $\exists x, y(x \lhd y \wedge \mathsf{nasal}(x) \wedge \mathsf{voiceless}(y))$ is true when $x = 3$ and $y = 4$. Hence, $^*\mathsf{NT}$ evaluates to false. On the other hand, in the model of the word *moon*, every value assigned x and y results in the sentence $\exists x, y(x \lhd y \wedge \mathsf{nasal}(x) \wedge \mathsf{voiceless}(y))$ evaluating to false. Hence the sentence $^*\mathsf{NT}$ evaluates to true and so $\mathcal{M}_{moon} \models {}^*\mathsf{NT}$.

This section has presented the first CDL: FO with successor. The FO with successor model has been studied carefully and it is known precisely what kinds of constraints can and cannot be expressed with this CDL (Thomas, 1982), as will be discussed below.

2.5 Feature-based Word Models

One way in which the successor model above is strange from a phonological perspective is its absence of phonological features. The properties associated with the elements of the domain are whole segments. However, nothing in model theory itself prohibits domain elements from having more than one property. It is a consequence of the construction in Table 2.8 that each domain element will satisfy exactly one of the unary relations a, no more and no less. We can formalize this statement of the successor model in Remark 1 as follows.

Remark 1 (The successor model entails disjoint unary relations)**.** For all successor models $\mathcal{M} = \langle D \mid (\mathsf{a})_{a\in\Sigma}, \lhd \rangle$, and for all $\mathsf{a},\mathsf{b} \in (\mathsf{a})_{a\in\Sigma}$, it is the case that $\mathsf{a} \cap \mathsf{b} = \varnothing$.

Therefore it is possible to design different models of words, where the unary relations do not represent segments like *a, b*, or *n* but phonological features such as *vocalic, labial*, or *nasal*. Crucially, in these models would not entail disjoint unary relations: a domain element could be both *voiced* and *labial* for instance.

 © Jeffrey Heinz

DRAFT

In this part of the chapter, we give one example of such a model. There are many others, as many as there are theories of phonological features. The model we give here is primarily for pedagogical reasons; we are not stating particular beliefs or arguments regarding the nature of feature systems. We are only choosing a simple system that illustrates some key points.

We set up a feature system with **privative** features for the simple alphabet Σ discussed earlier *a,b,d,e,g,h,i,k,l,m,n,o,p,r,s,t,u,z*. The use of privative features contrasts with the typical assumption in phonological theory that features are binary. We choose not to pick a minimal nor maximal set of features for distinguishing this set. Instead we choose somewhat arbitrarily a middle ground based on standard descriptive phonetic terms used for describing the manner, place and laryngeal qualities in articulating sounds. We call this model the "successor model with features." Its signature is shown below.

$$\langle D \mid \texttt{vocalic}, \texttt{low}, \texttt{high}, \texttt{front}, \texttt{stop}, \texttt{fricative}, \texttt{nasal}, \texttt{lateral} \\ \texttt{rhotic}, \texttt{voiced}, \texttt{voiceless}, \texttt{labial}, \texttt{coronal}, \texttt{dorsal}, \triangleleft \rangle \quad (2.4)$$

Table 2.4 shows how to construct a successor model with features for any string in Σ^*. Again this model ensures that distinct strings from Σ^* have different models and that every string has some model.

Figure 2.2 shows the successor model with features of the word *tent*.

The successor model with features contrasts sharply with the successor model with features in an important way. To see how, first consider the constraint *NT. Under the successor model with features, this constraint would be defined as in Example 2.5

Example 5 (The constraint *NT defined under the FO with successor model with features.)**.**

$$*\mathsf{NT} \stackrel{\text{def}}{=} \neg\exists x, y(x \triangleleft y \wedge \texttt{nasal}(x) \wedge \texttt{voiceless}(y)) \quad (2.5)$$

This looks similar to the definition of *NT under the successor model (Example 2.1), but there is a critical difference. The predicates above in Example 2.5 are *atomic* formula and not user-defined predicates as they are in Example 2.1.

This is an important ontological difference between these two models. In the successor model with features there is no primitive representational concept that corresponds to a sound segment like t like there is in the successor

 © Jeffrey Heinz

DRAFT

D	$\stackrel{\text{def}}{=}$	$\{1, 2, \ldots n\}$
vocalic	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{a, e, i, o, u\}\}$
low	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i = a\}$
high	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{i, u\}\}$
front	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{e, i\}\}$
stop	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{b, d, g, k, p, t\}\}$
fricative	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{h, s, z\}\}$
nasal	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{m, n\}\}$
lateral	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i = l\}$
rhotic	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i = r\}$
voiced	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{b, d, g, z\}\}$
voiceless	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{k, p, s, t, h\}\}$
labial	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{b, p, m\}\}$
coronal	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{d, s, t, z\}\}$
dorsal	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i \in \{d, g, k\}\}$
$\triangleleft$	$\stackrel{\text{def}}{=}$	$\{(i, i+1) \mid 1 \leq i < n\}$

Table 2.4: Creating a successor model with features for any word $w = a_1 a_2 \ldots a_n$.

model without features. Conversely, in the successor model without features there is no primitive representational concept that corresponds to a phonological like *voiceless* like there is in the successor model with features. In the successor model with features we can write user-defined predicates that define properties of domain elements that we can interpret to mean "being t".

$$\mathsf{is\_t}(x) \stackrel{\text{def}}{=} \texttt{stop}(x) \wedge \texttt{coronal}(x) \wedge \texttt{voiceless}(x) \tag{2.6}$$

Other sound segments would be defined similarly.

One way to put this difference is that in the successor model with features one can immediately determine whether a domain element is voiced or not, but in the successor model without features one cannot immediately determine this fact. Instead one can deduce it by checking the appropriate user-defined predicate. Likewise, in the successor model with features one cannot immediately determine whether a domain element is t or not. With

 © Jeffrey Heinz

DRAFT

$$\mathcal{M}_{tent} =$$

$$\Big\langle \{1,2,3,4\} \mid \begin{aligned} \texttt{vocalic} &= \{2\},\\ \texttt{front} &= \{2\},\\ \texttt{stop} &= \{1,4\},\\ \texttt{nasal} &= \{3\},\\ \texttt{voiceless} &= \{1,4\},\\ \texttt{coronal} &= \{1,3,4\}\\ \lhd &= \{(1,2),(2,3),(3,4)\} \end{aligned} \Big\rangle$$

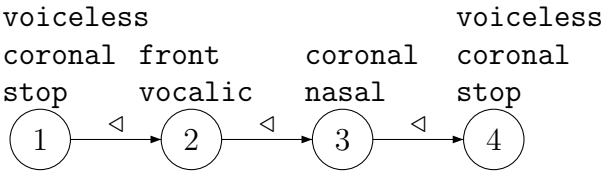

Figure 2.2: At left, the successor model with features of the word *tent*. Unary relations which equal the empty set are omitted for readability. At right, a graphical diagram of this model.

the featural representations, such a fact must be deduced with a user-defined predicate like the one above.

Also, the fact that such user-defined predicates exist should not be taken for granted. They exist here because the only logical system discussed so far is FO. With FO logic, it is possible to define a predicate for any subset of the alphabet Σ for both successor models with and without features. If the logical system was restricted in some further way then some user-defined predicates may not be possible to define. For example, if the logical system only permitted conjunction and no other Boolean connective then it would not be possible to define a predicate for voiceless stops in the successor model without features. This interplay between representations and logical power with respect to expressivity is an important theme of this chapter. It will be discussed at length with respect to the successor relation, and we will return to it in the context of features when restricted logics are introduced towards the end of the chapter.

As a consequence of FO logic then, any constraint definable with one of the representations discussed so far is definable in the other. This leads to the conclusion that there are no typological distinctions between the FO with successor theory and the FO with successor with features theory. Both admit exactly the same class of constraints.

However, while the two models do not make different typological predic-

DRAFT

tions, they do make different psychological ones. In regard to phonological theory, the signature of the model is an ontological commitment to the psychological reality of the model vocabulary. Taken seriously, the successor model with features says that the mental representations of words carries only the information shown in Figure 2.2. Thus, taken seriously, the successor model with features says that the segments in the word *tent* are not perceived as such but are instead perceived in terms of their features. Clever psycholinguistic experiments might be able to bring evidence to bear on which model more accurately resembles them actual mental representations of words.

2.6 Monadic Second-Order Logic

DRAFT

This section introduces Monadic Second-Order (MSO) logic. This logic is strictly *more expressive* than FO logic. We motivate the discussion of MSO logic from a linguistic perspective by showing that FO with successor, both with and without features, is not sufficient to account for long-distance phonotactic constraints.

What are long-distance phonotactic constraints? Odden (1994) draws attention to an unbounded nasal assimilation in Kikongo whereby underlying /ku-kinis-il-a/ becomes [kukinisina] 'to make dance for.' From one perspective, this assimilation could be said to be driven by a phonotactic constraint that forbids laterals from occuring after nasals. Similar long-distance constraints have been posited for a variety of long-distance assimilation and dissimilation processes (Hansson, 2010).

We first show that the phonotactic constraint which bans laterals from occuring *anywhere* after nasals cannot be expressed in the FO with successor model. As we hope to make clear, the problem is that the notion of *precedence* is not FO-definable from successor. To illustrate, in Kikongo, [kukinisila] is an ill-formed string. The nasal has only one successor [i], but [n] *precedes* many segments including the second and third [i]s and the [s,l] and [a]. It is the fact that [n] precedes [l] which makes [kukinisila] ill-formed according to the phonotactic constraint which bans laterals from occuring *anywhere* after nasals. We refer to this constraint as *N..L.

Constraint *N..L is not FO definable with successor. To prove this we use an abstract characterization of the constraints definable with FO and successor due to Thomas (1982) and reviewed in Rogers and Pullum (2011).

 © Jeffrey Heinz

Theorem 1 (Characterization of FO-definable constraints with successor)**.** *A constraint is FO-definable with successor if and only if there are two natural numbers k and t such that for any two strings w and v, if w and v contain the same substrings x of length k* the same number of times counting only up to t*, then either both w and v violate the constraint or neither does.*

Essentially, this theorem says constraints that are FO-definable with successor cannot distinguish among strings that are composed of the same number and type of substrings of some length k, where substrings can be counted only up to some threshold t.

We can use this theorem to show that *N..L is not FO definable with successor by presenting two strings which *N..L distinguishes but which are not distinguishable according to the criteria in Theorem 1. This would prove that *N..L is not LTT and thus not FO-definable with successor. Importantly, for any k and t we have to present two strings. These strings can depend on k and t.

We use notation a^k to mean the string consisting of k consecutive as. So $a^3 = aaa$. For any numbers k and t larger than 0, consider the words $w = a^k n a^k \ell a^k$ and $v = a^k \ell a^k n a^k$. Table 2.6 below shows the substrings up to length k, and their number of occurrences. Each word has the same substrings and the same number of them. Note the left and right word boundaries ($\rtimes$ and $\ltimes$ respectively) are customarily included as part of the strings.

In the discussion below, the following concept will prove useful. For every number t and every number n let the t-number of n equal n if $n < t$ otherwise let it be t. So if n is our count than the t-number of n is just the count of n up to the threshold t.

As can be seen from the above table, the two strings have exactly the same number of occurrences of each k-long substring. Consequently, the t-numbers of each k-long substring is also the same for any t. It follows, from Theorem 1 that these two strings cannot be distinguished by any constraint which is FO-definable with successor. More precisely, *any constraint which is FO-definable with successor is unable to distinguish in strings w and v whether n precedes ℓ or whether ℓ precedes n.* As such, no FO-definable constraint with successor can be violated by w but not by v and vice versa. It follows that *N..L is not FO definable with successor because for the reason that it this is precisely the distinction it makes.

Having established that linguistically motivated long-distance phonotac-

DRAFT

 © Jeffrey Heinz

count	$w = \rtimes a^k n a^k \ell a^k \ltimes$	Notes
1	$\rtimes a^{k-1}$	
3	a^k	
1	$a^i n a^j$	(for each $0 \leq i, j \leq k-1$, $i + j = k - 1$)
1	$a^i \ell a^j$	(for each $0 \leq i, j \leq k-1$, $i + j = k - 1$)
1	$a^{k-1} \ltimes$	

count	$v = \rtimes a^k \ell a^k n a^k \ltimes$	Notes
1	$\rtimes a^{k-1}$	
3	a^k	
1	$a^i n a^j$	(for each $0 \leq i, j \leq k-1$, $i + j = k - 1$)
1	$a^i \ell a^j$	(for each $0 \leq i, j \leq k-1$, $i + j = k - 1$)
1	$a^{k-1} \ltimes$	

Table 2.5: The k-long substrings with their number of occurrences in the strings $w = a^k n a^k \ell a^k$ and $v = a^k \ell a^k n a^k$ with word boundaries.

DRAFT

tic constraints are not FO-definable with successor, we turn to the question of how such constraints can be defined from the logical perspective offered here. Essentially, there are two approaches. One is to increase the power of the logic. The other is to change the model—the representation—of strings. This section examines the first option and the next section examines the second option. This interplay between logical power and representations and how it affects the expressivity of the linguistic system is a running theme of this book.

Monadic Second Order (MSO) logic is a logical language that is strictly more powerful than FO logic. Constraints that are MSO-definable with successor include every constraint which is FO-definable with successor because every sentence and formula in FO logic with successor is also a sentence and formula in MSO logic with successor and is interpreted in the same way. In addition to first order variables, MSO comes with **second order variables**. Generally, variables that are second order are allowed to vary over n-ary relations. The restriction to monadic second order variables means the variables in this logic can only vary over unary relations, which corresponds to *sets* of domain elements. This contrasts with first order variables, which recall vary only over elements of the domain.

MSO logic is defined formally in the appendix to Part I, so here we

 © Jeffrey Heinz

introduce it informally with examples. In MSO logic, the MSO variables are expressed with capital letters such as X, Y, and Z to distinguish them from first order variables which use lowercase letters like x, y, and z. Observe that $x \in X$ and $X(x)$ are synonyms. As with first order variables, second order variables are introduced into sentences and formula with quantifiers.

Additional Symbols in MSO logic	
X, Y, Z	variables which range over sets of elements of the domain
$x \in X$	checks whether an element x belongs to a set of elements X
$X(x)$	checks whether an element x belongs to a set of elements X

Table 2.6: Together with the symbols of FO logic shown in Table 2.2, these symbols make up MSO logic.

With MSO logic over successor, it is now possible to define the precedence relation as shown below.

$$\mathsf{closed}(X) \stackrel{\text{def}}{=} (\forall x, y)\big[(x \in X \wedge x \lhd y) \rightarrow y \in X\big] \tag{2.7}$$

$$x < y \stackrel{\text{def}}{=} (\forall X)\big[(x \in X \wedge \mathsf{closed}(X) \rightarrow y \in X\big] \tag{2.8}$$

Intuitively, a set of elements X in the domain of a model of some word w satisfies $\mathsf{closed}(X)$ only if every successor of every element in X is also in X. In short, $\mathsf{closed}(X)$ is true only for sets of elements X which are transitively closed under successor. Then x precedes y only if for every closed set of elements X which x belongs to, y also belongs to X.

Figure 2.3 below illustrates these ideas. The successor model for the string *alaana* is shown. Six ellipses are shown, which represent the six nonempty sets of domain elements which are closed under successor and thus satisfy $\mathsf{closed}(X)$.

We can conclude that ℓ precedes n because every closed set which element 2 (which corresponds to ℓ) belongs to (X_1 and X_2) also includes the element 5 (which corresponds to n). Similarly, we can conclude that n does not precede ℓ because it is not the case that all closed sets which contain element 5 (which corresponds to n) also include element 2 (which corresponds to ℓ). Set X_4 for instance contains element 5 but not element 2.

Once the binary relation for precedence ($<$) has been defined, it is now straightforward to define the constraint *N..L with features.

$$\mathsf{*N..L} \stackrel{\text{def}}{=} \neg(\exists x, y)[x < y \wedge \mathtt{nasal}(x) \wedge \mathtt{lateral}(y)] \tag{2.9}$$

 © Jeffrey Heinz

DRAFT

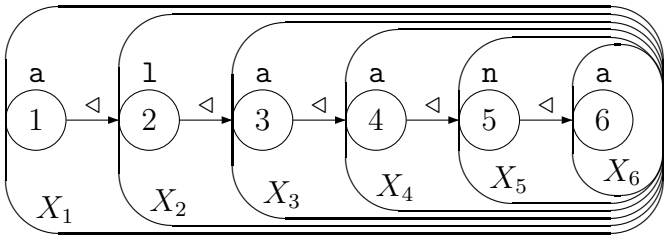

Figure 2.3: The successor model for the word *alaana*. Rectangular regions indicate the sets of domain elements (X_i) which are closed under successor.

The sentence above may look like a sentence of FO logic since no second order variables are present. However, it is important to remember that the precedence relation ($<$) is just an abbreviation for a longer formula, which is defined in MSO logic, and not within FO logic. Often whether a predicate is atomic or derived is not something that can be determined from inspecting a sentence or formula since the notation does not distinguish them. Usually one must be being acutely aware of the model signature to know whether a predicate is atomic or derived.

At this point, we have established that the linguistically motivated long-distance phonotactic constraint is not definable with FO logic with successor but it is definable with MSO logic with successor. We thus ask: What other kinds of constraints are MSO-definable with successor?

Another constraint that is not FO-definable with successor but is MSO-definable constraint with successor is a constraint that requires words to have an even number of nasals. Words like *man* and *neonatology* obey this constraint since they have two nasals but words like *mannequin* and *nanotechnology* do not since they have three nasals.

To see that this constraint is not FO-definable with successor, we use Theorem 1 as before. For any nonzero numbers k and t, consider the words $w = a^k(na^k)^{2t}$ and $v = a^k(na^k)^{2t}na^k$. Observe that w obeys the constraint since it contains $2t$ nasals and $2t$ is an even number. On the other hand, v contains $2t + 1$ nasals and therefore violates the constraint. However, as Table 2.7 shows, these words have the same substrings of length k, and the same t-numbers of each substring.

However, this constraint is expressible with MSO logic with successor. We make use of some additional predicates, including general precedence ($<$) defined in Equation 2.8. The predicate firstN is true of x only if x is the

DRAFT

$w = \rtimes a^k(na^k)^{2t}a^k \ltimes$			
count	t-number	k-long substrings	notes
1	1	$\rtimes a^{k-1}$	
$2t+2$	t	a^k	
$2t+2$	t	$a^i n a^j$	(for each $0 \leq i,j \leq k-1$, $i+j=k-1$)
1	1	$a^{k-1}\ltimes$	

$v = \rtimes a^k(na^k)^{2t}na^k \ltimes$			
count	t-number	k-long substrings	notes
1	1	$\rtimes a^{k-1}$	
$2t+2$	t	a^k	
$2t+2$	t	$a^i n a^j$	(for each $0 \leq i,j \leq k-1$, $i+j=k-1$)
1	1	$a^{k-1}\ltimes$	

Table 2.7: The k-long substrings and the t-numbers of their counts in $w = a^k(na^k)^{2t}$ and $v = a^k(na^k)^{2t}na^k$ with word boundaries.

first nasal occuring in the word (Equation 2.10). The predicate lastN is true of x only if x is the last nasal occuring in the word (Equation 2.11). Also, two variables x and y stand in the $\lhd_{\mathsf{N}}$ only if y is the first nasal to occur after x (Equation 2.12). So $\lhd_{\mathsf{N}}$ is a a successor relation relativized to nasals.

$$\mathsf{firstN}(x) \stackrel{\text{def}}{=} \mathtt{nasal}(x) \wedge \neg(\exists y)[\mathtt{nasal}(y) \wedge y < x] \tag{2.10}$$

$$\mathsf{lastN}(x) \stackrel{\text{def}}{=} \mathtt{nasal}(x) \wedge \neg(\exists y)[\mathtt{nasal}(y) \wedge x < y] \tag{2.11}$$

$$x \lhd_{\mathsf{N}} y \stackrel{\text{def}}{=} \mathtt{nasal}(x) \wedge \mathtt{nasal}(y) \wedge x < y \wedge \neg(\exists z)[\mathtt{nasal}(z) \wedge x < z < y] \tag{2.12}$$

Note we use the shorthand $x < y < z$ for $x < z \wedge z < y$.

With these predicates in place, we write Even-N as in Equation 2.13.

$$\begin{aligned} \textsc{Even-N} \stackrel{\text{def}}{=} (\exists X)\Big[& (\forall x)[\mathsf{firstN}(x) \rightarrow X(x)] \\ & \wedge (\forall x)[\mathsf{lastN}(x) \rightarrow \neg X(x)] \Big] \\ & \wedge (\forall x, y)\big[x \lhd_{\mathsf{N}} y \wedge \big(X(x) \leftrightarrow \neg X(y)\big)\big] \end{aligned} \tag{2.13}$$

In English, this says that a model of word w satisfies Even-N provided there is a set of domain elements X that includes the first nasal (if one

 © Jeffrey Heinz

DRAFT

occurs), does not include the last nasal (if one occurs) and for all pairs of successive nasals (if they occur), exactly one belongs to X. Consequently, words containing zero nasals satisfy the EVEN-N because the empty set of domain elements vacuously satisfies these three conditions. Words containing exactly one nasal do not satisfy EVEN-N because the first nasal and the last nasal are the same element x and it cannot both belong and not belong to X. However, words with exactly two nasals do satisfy EVEN-N because the first nasal belongs to X (satisfying the first condition), the last nasal does not (satisfying the second condition), and these two nasals are successive nasals and so are subject to the third condition, which they satisfy because exactly one of them (the first nasal) belongs to X. A little inductive reasoning along these lines lets one conclude that only words with an even number of nasals will satisfy EVEN-N as intended.

It is natural to wonder whether there is an abstract characterization of constraints that are MSO-definable with successor in the same way that Thomas (1982) provided an abstract characterization of constraints that are FO-definable with successor. In fact there is. Büchi (1960) showed that these constraints are exactly the ones describable with finite-state automata.

Theorem 2 (Characterization of MSO-definable constraints with successor)**.** *A constraint is MSO-definable with successor if and only if there is a finite-state automata which recognizes the words obeying the constraint.*

From the perspective of formal language theory, they are exactly the regular languages. Informally, these are formal languages for which the membership problem can be solved with a constant, finite amount of memory.

In this section we showed that FO-definable constraints with successor are not sufficiently powerful to express long-distance phonotactic constraints. One approach is to then increase the power of the logic. One logical system extends FO by adding quantification over monadic second order variables. This logic—MSO logic with successor—is able to express long-distance phonotactic constraints. However, MSO logic with successor also is also sufficiently expressive as a CDL to express constraints like EVEN-N.

Another way of putting it is like this. In the successor model, the information that in the word *alaana* the ℓ precedes the n is not immediately available from the representation. That information can be *deduced* but the deduction requires some computational effort. From the logical perspective taken here, this deduction requires MSO power and not FO power. Furthermore, once

DRAFT

 © Jeffrey Heinz

MSO power is admitted then it becomes possible to similarly deduce whether or not there are even numbers of elements with certain properties.

Another approach to developing a CDL which can express long-distance phonotactic constraints but not Even-N is to change the representation of strings; that is, to change the model signature. This is precisely the topic of the next section.

2.7 The Precedence Word Model

So far, the logics we have considered have been defined with respect to the successor model of words. However, as we have seen with phonological features vis a vis atomic letters, there are different models of strings. In this section, we consider the *precedence* model of strings. Simply, this model contains the precedence relation instead of the successor relation in its signature.

As with the successor model, there is a general construction for the determining the precedence model for any string. Given a string of w of length n the precedence model is constructed as follows. Since w is a sequence of n symbols, we let $w = a_1 a_2 \dots a_n$. Then set the domain $D = \{1, 2, \dots n\}$. For each symbol $a \in \Sigma$ and i between 1 and n inclusive, $i \in \mathtt{a}$ if and only if $a_i = a$. And finally, for each i and j between 1 and n inclusive, the only elements of the precedence relation are (i, j) so long as $i < j$. This is summarized in Table 2.8. This construction guarantees the model's soundness: each string

D	$\stackrel{\text{def}}{=}$	$\{1, 2, \dots n\}$
$\mathtt{a}$	$\stackrel{\text{def}}{=}$	$\{i \in D \mid a_i = a\}$ for each unary relation $\mathtt{a}$
$<$	$\stackrel{\text{def}}{=}$	$\{(i, j) \subseteq D \times D \mid i < j\}$

Table 2.8: Creating a successor model for any word $w = a_1 a_2 \dots a_n$.

has a model and distinct strings will have distinct models.

Figure 2.4 shows the precedence model for the word *tent* in addition to a graphical diagram of it on its right.

The difference between the precedence model and the successor model is how the order of segments in the word are represented. In the precedence model, the fact that the n is preceded by t in the word *tent* is immediately available because the element corresponding to t is in the precedence relation with the element corresponding to the first t. Under the successor model,

 © Jeffrey Heinz

DRAFT

$$
\begin{aligned}
&\mathcal{M}_{tent} \\
&= \langle D \mid \mathtt{t}, \mathtt{e}, \mathtt{n}, \mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}, < \rangle \\
&= \Big\langle \{1,2,3,4\} \mid \{1,4\}, \{2\}, \{3\}, \\
&\qquad \varnothing, \varnothing, \ldots \varnothing, \\
&\qquad \{(1,2),(1,3),(1,4), \\
&\qquad\quad (2,3),(2,4),(3,4)\} \Big\rangle
\end{aligned}
$$

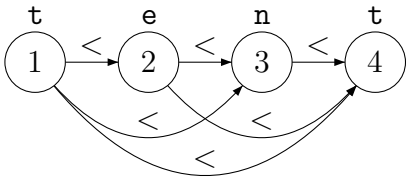

Figure 2.4: At left, the precedence model of the word *tent*. At right, a graphical diagram of this model.

this information was not immediately available as it was not part of the representation. However, under the precedence model it is.

Take seriously from a psychological perspective, the precedence model can be taken to mean that as words are perceived, information about the precedence relations is being stored in memory as part of the lexical representation of the word.

Also, in the same way that we considered the successor model both with and without features, we can also consider a precedence model with and without features. The precedence model introduced above was without features, but it is a simple matter to replace the unary relations in that model with the ones in Table 2.4.

It is straightforward to now write the constraint *N..L in the CDL which we call "FO with precedence with features."

$$\mathtt{*N..L} \stackrel{\text{def}}{=} \neg\exists x, y(x < y \wedge \mathtt{nasal}(x) \wedge \mathtt{lateral}(y)) \tag{2.14}$$

Equation 2.14 looks identical to Equation 2.9. However, there is critical difference. In Equation 2.14, the precedence relation is an atomic formula but in Equation 2.9 it is a user-defined predicate in MSO logic.

It is natural to ask of course whether a constraint like *NT is expressible in this CDL. The answer is Yes because successor is FO-definable from precedence. Equation 2.15 shows how. Essentially, x is succeeded by y only if x precedes y and there is no element z such that $z < y$ and $x < z$.

$$x \lhd y \stackrel{\text{def}}{=} x < y \wedge \neg(\exists z)[x < z < y] \tag{2.15}$$

It is a striking fact that successor is FO-definable from precedence but

DRAFT

precedence is MSO-definable from successor. This is a considerable asymmetry between the successor and precedence models of strings.

There are two important consequences. The first is the CDL "FO with precedence" properly subsumes the CDL "FO with successor." Not only is every constraint expressible with the CDL "FO with successor" also expressible with the CDL "FO with precedence", but there are constraints like `*N..L` above that expressible with the CDL "FO with precedence" but not with the CDL "FO with successor."

Another important consequence is that the CDL "MSO with precedence" is equivalent in expressive power to the CDL "MSO with successor" discussed in the previous section. This is because with MSO logic, precedence can be defined from successor as shown previously. Likewise because MSO logic properly extends FO logic, successor can also be defined from precedence. So at the level of MSO, these two models make no distinctions among the kinds of constraints that can be expressed. Constraints in each CDL correspond to exactly the regular stringsets.

There is also an abstract characterization of the FO-definable constraints with precedence due to McNaughton and Papert (1971).

Theorem 3 (Characterization of FO-definable constraints with precedence)**.** *A constraint is FO-definable with precedence if and only if there is a poistive integer n such that for all strings x, y, z if xy^nz obeys the constraint then for all $k > n$, xy^kz obeys the constraint too.*

This characterization says that FO-definable constraints with precedence can only distinguish iterations within strings up to some finite n. Two strings xy^iz and xy^jz with both $i, j > n$ but $i \neq j$ cannot be distinguished by any FO-definable constraint with precedence. As McNaughton and Papert (1971) amply document, there are other independently-motivated characterizations of this class as well.

The above characterization can be used to show that Even-N is not FO-definable with precedence. Again, the strategy is to consider any n and then to find strings w, v, x, y, z and numbers $i, j > n$ such that $w = xy^iz$ and $v = xy^jz$ where Even-N distinguishes w and v in the sense that one violates Even-N and the other does not. If the constraint were FO-definable with precedence such strings could not exist by Theorem 3. In this case, one solution is to set $x = z = \lambda$ (the empty string), $y = ma$, $i = 2n$ and $j = 2n + 1$. Then $w = (ma)^{2n}$ and $v = (ma)^{2n+1}$. Clearly, w has an even number of nasals since it has $2n$ [m]s but v has an odd number since it has

 © Jeffrey Heinz

DRAFT

$2n+1$ [m]s. Thus Even-N distinguishes these strings and thus by Theorem 3 it cannot be FO-definable with precedence.

In this section, we considered a model of words where order is represented with the precedence relation instead of the successor relation. It was shown that long-distance constraints can readily be expressed in the CDL "FO with precedence." Furthermore, local phonotactic constraints like *NT can also be expressed because successor is FO-definable from precedence. However, the converse is not true. This asymmetry means that FO with precedence is strictly more expressive than "FO with successor." It was also shown that Even-N is not expressive in this system. Finally, it was noted that "MSO with precedence" is equally expressive as "MSO with successor". Once there is MSO power, successor and precedence are each definable from the other. Which constraints can be expressed by which CDLs is summarized in Figure 2.5.

DRAFT

MSO	*N..L, Even-N	Even-N
FO	*NT	*NT, *N..L
	$\lhd$	$<$

Figure 2.5: Classifying the constraints *NT, *N..L, and Even-N.

More generally, this section established the following. Although one way to increase the expressivity of a CDL is to increase the power of the logic, another way is to change the representations underlying the models. This speaks directly to the interplay between representations and computational power, one of the themes of this chapter.

We conclude that the only CDL discussed so far that can express both local and long-distance phonotactic constraints (like *NT and *N..L) and fails to express constraints like Even-N is the CDL "FO with precedence."

 © Jeffrey Heinz

2.8 Discussion

2.8.1 Tradeoffs between representations and power

2.8.2 Typology, learnability, and psychological reality

2.8.3 Well-formedness and Transformations

2.9 Further Reading

DRAFT

 © Jeffrey Heinz

Chapter 3

Transformations, Logically

Jeffrey Heinz

DRAFT

This chapter explains how transformations from one representation to another can be described with the same logical tools introduced in the last chapter. Transformations are a central component of phonological theory, which posits a transformation exists between the abstract mental representations of the pronunciation of morphemes (the underlying form) to the more concrete, more directly observable, surface representation (the surface form). The mathematical and computational basis for this work is summarized in Courcelle (1994).

This chapter aims to introduce these ideas in an accessible way to linguists with a basic knowledge of phonology. However, the techniques have application beyond the theory of phonology to any other subfield of linguistics, notably morphology and syntax, in part because these methods apply equally well to trees and graphs, not just strings. Also this chapter is merely an introduction to these methods. As such, it introduces them in the context of string-to-string transformations; that is, *functions* from strings to strings. As a matter of fact, these methods have been generalized by computer scientists to describe *weighted relations* between strings . These generalizations permit one to describe and characterize optionality and exceptionality, in addition to gradient and probabilistic generalizations. Such generalizations are not discussed until the end of the book (Chapter XYZ) for two reasons. First they are unnecessarily complicate the central ideas, which are easier to first understand without them. Second, much valuable work can be done without them, as exemplified by the chapters in parts 2 and 3 of this book.

The application of these methods for phonological description and theory is what primarily distinguishes this work from One-Level Declarative Phonology developed by Bird, Coleman, and Scobbie twenty five years ago. That research, like the research in this book, emphasizes the importance of a declarative approach to phonological description and theory. The key difference is that twenty-five years ago transformations were studied within "one level." In other words, transformations were understood as *constraints* on unspecified representations. As such, those 'transformations' could only *add* (further specify) information to representations. In contrast, in this chapter we will see how logic can be used to literally add, subtract, change, or more generally *transform* one representation into another. For this reason, one could say that the Computational Generative Phonology apperoach in this book is essentially a form of *Two-level* Declarative Phonology.

DRAFT

3.1 Strings-to-string Transformations

A logical description of a string-to-string function uses logic to explain how to an input string is mapped to an output string. As with the constraints in the previous chapter, the logic does not operate over the strings themselves, but over the model-theoretic representation of those strings. Therefore, a logical description of a string-to-string function uses logic to convert an input *model* of a string into another *model* (of a possibly different string). Recall that the model of a string is understood in terms of its *signature*. The signature includes the relations over the domain of the model that must be specified in order to uniquely identify some string. Therefore, the logical description needs to specify the domain and relations in the *output model* in terms of the the domain and relations in the *input model*.

A logical description of a function specifies the domain of the function, and for each input, it must specify the domain of the output model and the relations over it with logical formulas, interpreted with respect to the input model.[1] The domain of the function is specified by the *domain formula*. The domain of the output model is specified by three ingredients: the *copy set*

[1]Note there are two distinct meanings of the word 'domain' in use here. The first has to do with the domain of a *function* and the second with the domain of a *model*. A function's domain is the set of elements over which the function is defined. For instance for $F : A \to B$, the domain is the set A. In contrast, the domain of a model is the elements in the 'universe' the model is describing. In finite model theory, which is used in this book, the domain of a model is a finite set of natural numbers $1,\ldots n$, representing the finitely

 © Jeffrey Heinz

and the *licensing formula*. The relations over the output model are specified by *relational formulas* for each of the relations in the signature of the output model. These formulas are evaluated with respect the *input model* in a way that will be made clear below. In our first examples, we leave out the copy set and return to it in section 3.2.

3.1.1 Word-final obstruent devoicing

For concreteness, let us provide a logical description of the phonological process of word-final obstruent devoicing. This process maps strings with word-final voiced obstruents to voiceless ones. For example, this process maps the string *hauz* to *haus* and the string *bad* to *bat*. Words without word-final voiced obstruents surface faithfully so this process also maps the string *haus* to *haus*.

We choose to model this process with the feature-based successor model described in 2.5 (see Table 2.4). Strings in both the input and the output will be represented with the feature-based successor model. Note this is a choice. One can choose to model the input, the output, or both, will some other word model, such as the conventional word model with successor. It follows we want to provide a logical transformation which maps the model of *hauz* to the model of *haus*, as shown in Figure 3.1. We introduce the logical formulas one at a time and then summarize them at the end of the example.

We must specify the domain of the function f with a logical formula with no free variables $\phi_{\texttt{domain}}$. For a string w, the function f is defined if and only if its model $\mathcal{M}_w \models \phi_{\texttt{domain}}$. In this case, we want word-final obstruent devoicing to apply to every string. Hence we let $\phi_{\texttt{domain}} \stackrel{\text{def}}{=} \texttt{true}$.

How is the domain of the output model of $f(\mathcal{M}_w)$ determined? Logical transductions fix the domain of the output as a *copy* of the input domain. For example, as shown in Figure **??**, the domain of $\mathcal{M}_{hauz}$ is {1,2,3,4}. Therefore, the domain of $f(\mathcal{M}_{hauz})$ is also {1,2,3,4}. An immediate consequence is that it appears that functions cannot alter the size of the input upon which they are acting. However, it is precisely the copy set (section 3.2) and the licensing formula $\phi_{\texttt{license}}$ (3.1.2) which determines the size of the output model. Basically, the copy set allows transformations to relate *larger* outputs to *smaller* inputs and the licensing formula (section **??**) allows transformations to relate *smaller* outputs to *larger* inputs. Working together these ingredients

many elements in the universe.

 © Jeffrey Heinz

DRAFT

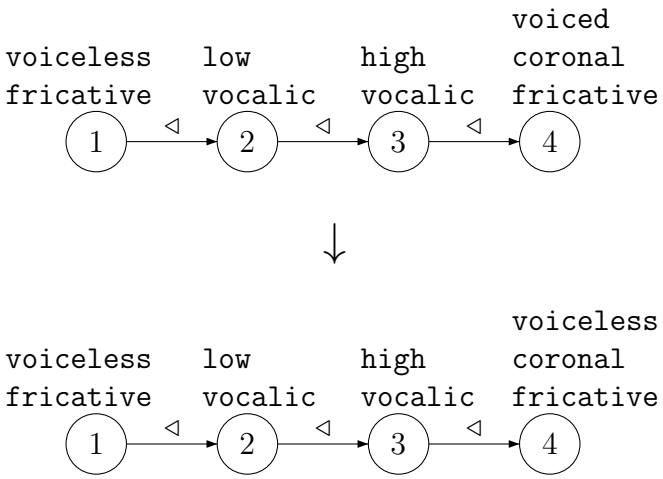

Figure 3.1: A graphical diagram of the feature-based successor model of *hauz* being mapped to the feature-based successor model of *haus*.

let one relate any input model of size n to an output model of size m, for any $n, m \in \mathbb{N}$. For now, since the word-final voiced obstruent devoicing *does preserve* the size of each input in the output, we set aside the copy-set and licensing formula for now.[2]

Thus given $\mathcal{M}_{hauz}$, f sets the domain of the output model to {1,2,3,4}. Finally, we must determine the relations which hold over these elements. For each relation R of arity n in the signature of output model, we must specify a formula $\phi_{\mathtt{R}}$ with n free variables $\phi_{\mathtt{R}}(x_1, \ldots x_n)$. Since the signature of the output model has one binary relation (the successor relation $\lhd$) and several unary features (the phonological features), we need a formula for each of these.

How are these formula interpreted? As follows. For any string-to-string

[2]For completeness, in this case the copy set $C = \{1\}$ and the $\phi_{\mathtt{license}} \stackrel{\text{def}}{=} \mathtt{true}$.

 © Jeffrey Heinz

DRAFT

function f and input model $\mathcal{M}_w$, the elements $x_1, \ldots x_n$ in the domain of the output model $f(\mathcal{M}_w)$ stand in the n-ary relation R in the output signature if and only if $\phi_{\mathtt{R}}(x_1, \ldots x_n) \models \mathcal{M}_w$.

For example, the successor relation is a binary relation in the output signature. So we must define $\phi_{\lhd}(x, y)$. Since word-final obstruent devoicing does not affect the successor relations, we define this function as follows.

$$\underbrace{\phi_{\lhd}(x,y)}_{\text{Do x and y in the output model stand in the successor relation?}} \stackrel{\text{def}}{=} \underbrace{x \lhd y}_{\text{Evaluate with respect to the input model.}}$$

This means the following: elements x and y *in the output* stand in the successor relation if and only if corresponding elements x and y *in the input* satisfy the successor relation in the input model. Since $1 \lhd 2$ in the input model, it follows that elements 1 and 2 likewise stand in the successor relation in the output model. Similarly, since elements 1 and 3 *do not stand* in the successor relation in the input model, it follows that they do not stand in the successor relation in the output model. Consequently, the formula above guarantees (in fact literally says) that the successor relation in the output will be the same as the successor relation in the input.

As another example, consider the unary relation `vocalic`. As this is a unary relation, we must define a formula with one free variable $\phi_{\mathtt{vocalic}}(x)$. Let us define it as follows.

$$\underbrace{\phi_{\mathtt{vocalic}}(x)}_{\text{Does x have the feature vocalic in the output model?}} \stackrel{\text{def}}{=} \underbrace{\mathtt{vocalic(x)}}_{\text{Evaluate with respect to the input model.}}$$

It follows from this definition that domain element x in the output model is vocalic if and only if the corresponding domain element x in the input is vocalic. Thus, as we expect word final obstruent devoicing does not affect the vocalic nature of elements within a string.

As we know, the only features affected by word-final devoicing are *voicing* features, which in our model are `voiced` and `voiceless`. All other unary relations in the signature of the output model will be defined similarly to $\phi_{\mathtt{vocalic}}(x)$ (as shown in Table 3.1 on page 57). However, the voicing features are affected by this process, so how do we specify which domain elements are voiced or voiceless? The voiced elements will be the ones that were

DRAFT

voiced in the input and are not the ones which are word-final obstruents. We can formalize this as follows. It will be useful to write some user-defined predicates.

$$\textsf{word-final}(x) \stackrel{\text{def}}{=} \neg\exists y\ (x \lhd y) \tag{3.1}$$

$$\textsf{obstruent}(x) \stackrel{\text{def}}{=} \texttt{stop}(x) \vee \texttt{fricative}(x) \tag{3.2}$$

We thus define $\phi_{\texttt{voiced}}(x)$ as follows.

$$\underbrace{\phi_{\texttt{voiced}}(x)}_{\substack{\text{Does x have the feature } \texttt{voiced} \\ \text{in the output model?}}} \stackrel{\text{def}}{=} \underbrace{\texttt{voiced(x)} \wedge \neg(\textsf{word-final(x)} \wedge \textsf{obstruent(x)})}_{\substack{\text{Evaluate with respect to the} \\ \text{input model.}}}$$

Similarly, the domain elements in the output which are voiceless are those that are voiceless in the input or those that are word-final obstruents.

$$\underbrace{\phi_{\texttt{voiceless}}(x)}_{\substack{\text{Does x have the feature } \texttt{voiceless} \text{ in the output} \\ \text{model?}}} \stackrel{\text{def}}{=} \underbrace{\texttt{voiceless(x)} \vee (\textsf{word-final(x)} \wedge \textsf{obstruent(x)})}_{\substack{\text{Evaluate with respect to the} \\ \text{input model.}}}$$

For completeness, we show the complete logical description of word-final devoicing.

3.1.2 Word-final vowel deletion

Let us consider another example, word-final vowel deletion, which will illustrate the role played by the licensing formula. Word-final vowel deletion has been argued to be a process in Yawelmani Yokuts. It essentially maps strings like *paka* to *pak* and *pilot* to *pilot*.

As before, the domain of this function is all strings and so $\phi_{\texttt{domain}} \stackrel{\text{def}}{=} \texttt{true}$. Also as before, the domain of the output model is a copy of the domain elements of the input model. However, these domain elements of the output model do not automatically exist in the output model; they must be *licensed* by a formula with one free variable called the licensing formula $\phi_{\texttt{license}}(x)$. In other words, the domain elements of the output model are really the *licensed* copies of the domain elements of the input model. Since word-final

 © Jeffrey Heinz

DRAFT

$\phi_{\texttt{domain}}$	$\stackrel{\text{def}}{=}$	$\texttt{true}$
C	$\stackrel{\text{def}}{=}$	$\{1\}$
$\phi_{\texttt{license}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{true}$
$\phi_{\triangleleft}(x, y)$	$\stackrel{\text{def}}{=}$	$x \triangleleft y$
$\phi_{\texttt{vocalic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{vocalic}(x)$
$\phi_{\texttt{low}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{low}(x)$
$\phi_{\texttt{high}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{high}(x)$
$\phi_{\texttt{front}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{front}(x)$
$\phi_{\texttt{stop}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{stop}(x)$
$\phi_{\texttt{fricative}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{fricative}(x)$
$\phi_{\texttt{nasal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{nasal}(x)$
$\phi_{\texttt{lateral}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{lateral}(x)$
$\phi_{\texttt{rhotic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{rhotic}(x)$
$\phi_{\texttt{labial}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{labial}(x)$
$\phi_{\texttt{coronal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{coronal}(x)$
$\phi_{\texttt{dorsal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{dorsal}(x)$
$\phi_{\texttt{voiced}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{voiced}(x) \wedge \neg(\textsf{word-final}(x) \wedge \textsf{obstruent}(x))$
$\phi_{\texttt{voiceless}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{voiceless}(x) \vee (\textsf{word-final}(x) \wedge \textsf{obstruent}(x))$

Table 3.1: The complete logical specification for word-final obstruent devoicing when the input and output string models are both the feature-based successor model.

vowels delete in this process, all domain elements which do not correspond to word-final vowels are licensed.

$$\underbrace{\phi_{\texttt{license}}(x)}_{\text{Does x belong to the domain of the output model?}} \stackrel{\text{def}}{=} \underbrace{\neg(\textsf{word-final}(x) \wedge \texttt{vocalic}(x))}_{\text{Evaluate with respect to the input model.}}$$

Since this process does not affect the phonological features in the string, each of the unary relations R in the signature of the output model can be

 © Jeffrey Heinz

DRAFT

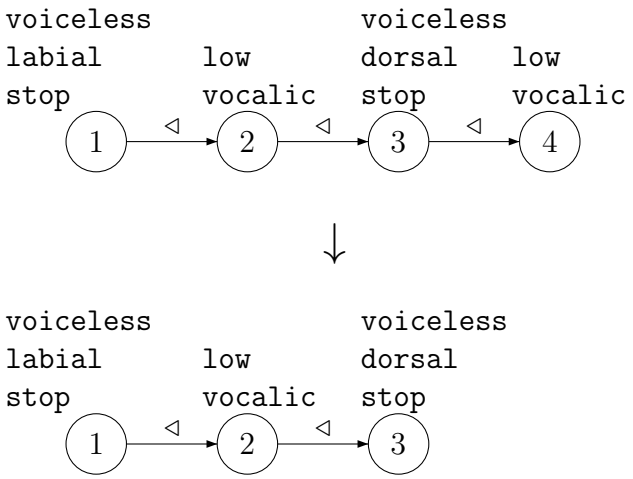

Figure 3.2: A graphical diagram of the feature-based successor model of *paka* being mapped to the feature-based successor model of *pak*.

defined as follows: $\phi_{\texttt{R}}(x) \stackrel{\text{def}}{=} \texttt{R}(x)$. In other words, $\phi_{\texttt{vocalic}}(x) \stackrel{\text{def}}{=} \texttt{vocalic}(x)$ and $\phi_{\texttt{voiced}}(x) \stackrel{\text{def}}{=} \texttt{voiced}(x)$ and so on. What about the binary successor relation? Letting $\phi_{\triangleleft}(x, y) \stackrel{\text{def}}{=} x \triangleleft y$ is sufficient. While it is true that $3 \triangleleft 4$ is true in the input, the fact that 4 is not licensed is sufficient to ensure that the pair $(3, 4)$ is not an element of the successor relation in the output model. The relations in the output model are always *restricted* to tuples which only contain *licensed* domain elements.

For completeness, Table 3.2 shows the complete logical description of word-final vowel deletion.

This section explained in more detail how the domain elements of the output model are determined. While these are always copies of the domain elements of the input model, it is not the case that every domain element in the input model is always copied as a domain element of the output model.

 © Jeffrey Heinz

DRAFT

$\phi_{\texttt{domain}}$	$\stackrel{\text{def}}{=}$	$\texttt{true}$
C	$\stackrel{\text{def}}{=}$	$\{1\}$
$\phi_{\texttt{license}}(x)$	$\stackrel{\text{def}}{=}$	$\neg(\textsf{word-final}(x) \wedge \texttt{vocalic}(x))$
$\phi_{\triangleleft}(x, y)$	$\stackrel{\text{def}}{=}$	$x \triangleleft y$
$\phi_{\texttt{vocalic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{vocalic}(x)$
$\phi_{\texttt{low}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{low}(x)$
$\phi_{\texttt{high}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{high}(x)$
$\phi_{\texttt{front}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{front}(x)$
$\phi_{\texttt{stop}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{stop}(x)$
$\phi_{\texttt{fricative}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{fricative}(x)$
$\phi_{\texttt{nasal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{nasal}(x)$
$\phi_{\texttt{lateral}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{lateral}(x)$
$\phi_{\texttt{rhotic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{rhotic}(x)$
$\phi_{\texttt{labial}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{labial}(x)$
$\phi_{\texttt{coronal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{coronal}(x)$
$\phi_{\texttt{dorsal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{dorsal}(x)$
$\phi_{\texttt{voiced}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{voiced}(x)$
$\phi_{\texttt{voiceless}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{voiceless}(x)$

Table 3.2: The complete logical specification for word-final obstruent devoicing when the input and output string models are both the feature-based successor model.

Only those elements x which satisfy $\phi_{\texttt{license}}(x)$ become domain elements in the output model.

3.2 Getting Bigger

So far we have exemplified logical transductions with phonological processes that change segmental material and processes that delete segmental material.

 © Jeffrey Heinz

DRAFT

How can logical transductions be used to define processes that *add* segmental material?

The answer to this question lies in the copy set. We have set aside this ingredient until now. In the previous examples, the copy set contained only *one* element. Thus each input element in the domain was copied exactly *once*. More generally, the copy set may contain n elements. It follows that the domain of the output model may contain n copies of *each* domain element of the input model. The copies of a domain element x in the input model are distinguished from each other using the names of the elements in the copy set. For example, consider the word *hauz* so that the domain elements of $\mathcal{M}_{hauz}$ are $\{1, 2, 3, 4\}$. If we are defining a logical transduction and define the copy set $C \stackrel{\text{def}}{=} \{1, 2\}$ then there are as many as *eight* domain elements in the output structure. It is customary to name these domain elements as *pairs*; the first coordinate indicates the domain element in the input model being copied and the second coordinate indicates *which* copy. Thus the pair $(1, 2)$ indicates the second copy of the first domain element of the input model and $(3, 1)$ indicates the first copy of the third element and so on. The eight possible domain elements in the output model are thus $\{(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2), (4, 1), (4, 2)\}$.

Whenever the copy set contains more than one element, the number of licensing formulas and relational formulas needed to describe the logical transduction multiplies as well. For each $i \in C$, there is a licensing formula $\phi^i_{\texttt{license}}(x)$. As before, this formula is evaluated with respect to the corresponding domain element in the input model. If it evaluates to true on x then the domain element (x, i) is licensed and belongs to the domain of the output model. Thus for a copy set C, there are $|C|$ licensing formulas.

Similarly, for each unary relation R in the signature of the output model, there are $|C|$ relational formulas: for each $i \in C$, we must define $R^i(x)$. The domain element (x, i) — the ith copy of x in the output model — belongs to R in the output model if and only if $R^i(x)$ evaluates to true in the input model.

For each binary relation R in the output signature, there are $|C|^2$ relational formulas $R^{i,j}(x, y)$ with $i, j \in C$. If and only if $R^{i,j}(x, y)$ evaluates to true with respect to the input model then the ith copy of x stands in the R relation to the jth copy of y in the output model. In which case, we have $((x, i), (y, j)) \in R$. If $R^{i,j}(x, y)$ evaluates to false with respect to the input model then $((x, i), (y, j))$ does not belong to R. For relations of higher arity,

 © Jeffrey Heinz

DRAFT

the licensing and relational formula multiply out similarly. Since the word models developed so far involve at most binary relations, we ignore relations of higher arity here (though they are discussed in the mathematical appendix **??**).

How the copy set works along with the additional formulas it entails are illustrated next with word-final vowel epenthesis and total reduplication. We provide complete logical descriptions of these transformations.

3.2.1 Word-final vowel epenthesis

Hindi speakers epenthesize the low vowel *a* to words which end in sonorant consonants (Shukla, 2000). We provide a logical description of this process given the the segments describable with the feature-based model. For example, this process would map the hypothetical word *pan* to *pana* as well as *pak* to *pak*. Figure 3.3 visualizes the mapping between the model structures *pan* and *pana*.

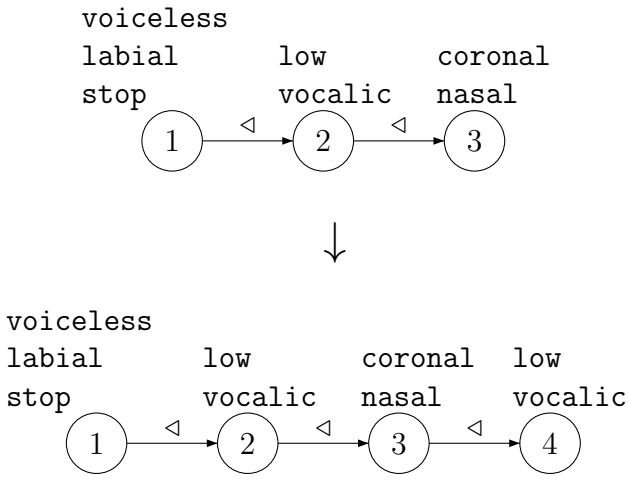

Figure 3.3: A graphical diagram of the feature-based successor model of *pan* being mapped to the feature-based successor model of *pana*.

First we can define sonorant consonants as follows.

$$\mathsf{sonorant\_C}(x) \stackrel{\text{def}}{=} \mathtt{nasal}(x) \vee \mathtt{lateral}(x) \vee \mathtt{rhotic}(x) \tag{3.3}$$

Next, we need a copy set of at least size 2 and so we define $C \stackrel{\text{def}}{=} \{1, 2\}$. Consequently, for the input *pan* which has three domain elements $\{1, 2, 3\}$,

 © Jeffrey Heinz

DRAFT

there are maximally 6 domain elements in the output model: $\{(1,1),(2,1),(3,1),(1,2),(2,2),(3,2)\}$. Since the copy set C has two elements, we must define two licensing formula, each with one free variable.

$$\phi^1_{\texttt{license}}(x) \stackrel{\text{def}}{=} \texttt{true} \tag{3.4}$$

$$\phi^2_{\texttt{license}}(x) \stackrel{\text{def}}{=} \mathsf{sonorant\_C}(x) \wedge \mathsf{word\text{-}final}(x) \tag{3.5}$$

$\phi^1_{\texttt{license}}(x)$ is always true so the first copy of each element is present. $\phi^2_{\texttt{license}}(x)$ is only true when $\mathsf{sonorant\_C}(x) \wedge \mathsf{word\text{-}final}(x)$ evaluates to true in the input model. For the word *pan* this occurs for $x = 3$, but for the word *pak* no x satisfies $\phi^2_{\texttt{license}}(x)$. Consequently, the output model of the process applied to *pan* has four domain elements $\{(1,1),(2,1),(3,1),(3,2)\}$ but the the output model of the process applied to *pak* has three domain elements $\{(1,1),(2,1),(3,1)\}$.

This is illustrated in Figure 3.4, where the first and second copies of the domain elements of *pan* are arranged in rows and the unlicensed elements are in gray.

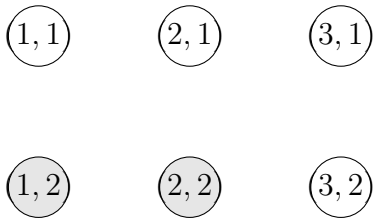

Figure 3.4: The possible domain elements of the output model for input *pan* when the copy set $C \stackrel{\text{def}}{=} \{1,2\}$. The unlicensed elements are colored gray.

Next, we turn to the binary successor relation in the output model. Here, we must have four formulas to specify the successor relation in the output signature. We define these as follows

$$\phi^{1,1}_{\lhd}(x,y) \stackrel{\text{def}}{=} x \lhd y \tag{3.6}$$

$$\phi^{1,2}_{\lhd}(x,y) \stackrel{\text{def}}{=} \mathsf{sonorant\_C}(x) \wedge \mathsf{word\text{-}final}(x) \wedge \mathsf{word\text{-}final}(y) \tag{3.7}$$

$$\phi^{2,1}_{\lhd}(x,y) \stackrel{\text{def}}{=} \texttt{false} \tag{3.8}$$

$$\phi^{2,2}_{\lhd}(x,y) \stackrel{\text{def}}{=} \texttt{false} \tag{3.9}$$

 © Jeffrey Heinz

DRAFT

Consequently, the successor relations are preserved among the first copy of the domain elements and the only successor from an element of the first copy to an element of the second copy is satisfied when x satisfies both $\mathsf{word\text{-}final}(x)$ and $\mathsf{sonorant\_C}(x)$.

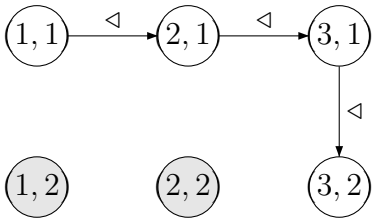

Figure 3.5: The successor relations in the output model for input *pan* when the copy set $C \stackrel{\text{def}}{=}= \{1, 2\}$. The unlicensed elements are colored gray.

Finally, we must define two formulas for each unary relation R in the output signature, $\phi^1_{\mathtt{R}}(x)$ and $\phi^2_{\mathtt{R}}(x)$. For each unary relation R, we define $\phi^1_{\mathtt{R}}(x) \stackrel{\text{def}}{=} R(x)$. Thus, the first copy of the domain elements are faithful to the unary relations they satisfied in the input. For the second copy, we can also let the domain elements be faithful to the unary relations they satisfied in the input with two exceptions. In our model,the low vowel *a* is low and vocalic and so $\phi^2_{\mathtt{vocalic}}(x)$ and $\phi^2_{\mathtt{low}}(x)$ must be defined to be true only when x corresponds to an element in the input that satisfies $\mathsf{sonorant\_C}(x)$ and $\mathsf{word\text{-}final}(x)$. For other unary relations R, we can define $\phi^2_{\mathtt{R}}(x) \stackrel{\text{def}}{=} \mathtt{false}$.

For completeness, Table 3.3 shows the complete logical description of word-final vowel epenthesis.

3.2.2 Duplication

Here we provide another example of a logical transduction, total reduplication. Obviously, we set the copy set $C \stackrel{\text{def}}{=} \{1, 2\}$. Then we essentially make all unary relations be faithful to their input so for all unary relations R in the output signature we have $\phi^1_{\mathtt{R}}(x) = \phi^2_{\mathtt{R}}(x) \stackrel{\text{def}}{=} R(x)$. As for the successor relation, two elements (x, i) and (y, j) stand in the successor relation if only if either $i = j$ and $x \triangleleft y$ in the input model or $i = 1$ and $j = 2$ and x is word-final in the input and y is word-initial in the input. We define $\mathsf{word\text{-}initial}(x)$ as follows.

$$\mathsf{word\text{-}initial}(x) \stackrel{\text{def}}{=} \neg\exists y\ (y \triangleleft x) \tag{3.10}$$

 © Jeffrey Heinz

DRAFT

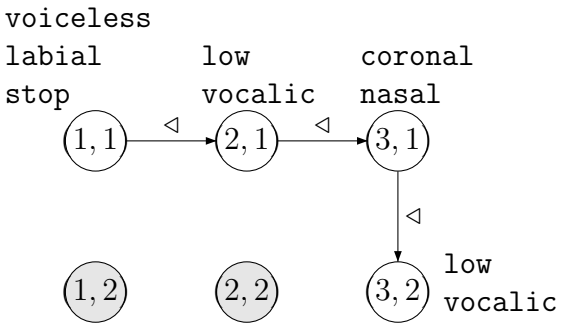

Figure 3.6: The model representing *pana* which is output for the input *pan*. The unlicensed elements are colored gray.

To illustrate, Figure 3.7 shows the output model for the input *pan*. In this way the copy set and the logical make it straightforward to define total reduplication.

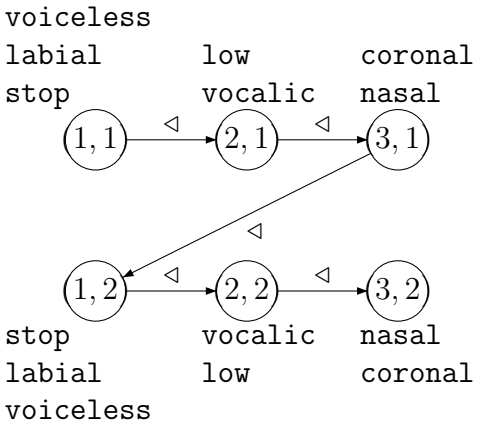

Figure 3.7: The model representing *panpan* is the output of the input *pan* to the process of reduplication.

For completeness, Table 3.4 shows the complete logical description of total reduplication.

 © Jeffrey Heinz

DRAFT

$\phi_{\texttt{domain}}$	$\stackrel{\text{def}}{=}$	$\texttt{true}$	C	$\stackrel{\text{def}}{=}$	$\{1,2\}$
$\phi^1_{\texttt{license}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{true}$	$\phi^2_{\texttt{license}}(x)$	$\stackrel{\text{def}}{=}$	$\mathsf{sonorant\_C}(x) \wedge \mathsf{word\text{-}final}(x)$
$\phi^{1,1}_{\triangleleft}(x,y)$	$\stackrel{\text{def}}{=}$	$x \triangleleft y$	$\phi^{1,2}_{\triangleleft}(x,y)$	$\stackrel{\text{def}}{=}$	$\mathsf{sonorant\_C}(x) \wedge \mathsf{word\text{-}final}(x) \wedge \mathsf{word\text{-}final}(y)$
$\phi^{2,1}_{\triangleleft}(x,y)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$	$\phi^{2,2}_{\texttt{succ}}(x,y)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{vocalic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{vocalic}(x)$	$\phi^2_{\texttt{vocalic}}(x)$	$\stackrel{\text{def}}{=}$	$\mathsf{sonorant\_C}(x) \wedge \mathsf{word\text{-}final}(x)$
$\phi^1_{\texttt{low}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{low}(x)$	$\phi^2_{\texttt{low}}(x)$	$\stackrel{\text{def}}{=}$	$\mathsf{sonorant\_C}(x) \wedge \mathsf{word\text{-}final}(x)$
$\phi^1_{\texttt{high}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{high}(x)$	$\phi^2_{\texttt{high}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{front}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{front}(x)$	$\phi^2_{\texttt{front}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{stop}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{stop}(x)$	$\phi^2_{\texttt{stop}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{fricative}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{fricative}(x)$	$\phi^2_{\texttt{fricative}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{nasal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{nasal}(x)$	$\phi^2_{\texttt{nasal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{lateral}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{lateral}(x)$	$\phi^2_{\texttt{lateral}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{rhotic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{rhotic}(x)$	$\phi^2_{\texttt{rhotic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{labial}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{labial}(x)$	$\phi^2_{\texttt{labial}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{coronal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{coronal}(x)$	$\phi^2_{\texttt{coronal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{dorsal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{dorsal}(x)$	$\phi^2_{\texttt{dorsal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{voiced}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{voiced}(x)$	$\phi^2_{\texttt{voiced}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$
$\phi^1_{\texttt{voiceless}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{voiceless}(x)$	$\phi^2_{\texttt{voiceless}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$

Table 3.3: The complete logical specification for word-final vowel epenthesis when the input and output string models are both the feature-based successor model.

3.2.3 Summary

At this point, we have covered how to define transformations logically. One remarkable aspect about these methods is that these can be used for different representations. We will see this is the case

DRAFT

$\phi_{\texttt{domain}}$	$\stackrel{\text{def}}{=}$	$\texttt{true}$	C	$\stackrel{\text{def}}{=}$	$\{1,2\}$
$\phi^1_{\texttt{license}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{true}$	$\phi^2_{\texttt{license}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{true}$
$\phi^{1,1}_{\triangleleft}(x,y)$	$\stackrel{\text{def}}{=}$	$x \triangleleft y$	$\phi^{1,2}_{\triangleleft}(x,y)$	$\stackrel{\text{def}}{=}$	$\textsf{word-final}(x) \wedge \textsf{word-initial}(y)$
$\phi^{2,1}_{\triangleleft}(x,y)$	$\stackrel{\text{def}}{=}$	$\texttt{false}$	$\phi^{2,2}_{\texttt{succ}}(x,y)$	$\stackrel{\text{def}}{=}$	$x \triangleleft y$
$\phi^1_{\texttt{vocalic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{vocalic}(x)$	$\phi^2_{\texttt{vocalic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{vocalic}(x)$
$\phi^1_{\texttt{low}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{low}(x)$	$\phi^2_{\texttt{low}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{low}(x)$
$\phi^1_{\texttt{high}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{high}(x)$	$\phi^2_{\texttt{high}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{high}(x)$
$\phi^1_{\texttt{front}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{front}(x)$	$\phi^2_{\texttt{front}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{front}(x)$
$\phi^1_{\texttt{stop}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{stop}(x)$	$\phi^2_{\texttt{stop}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{stop}(x)$
$\phi^1_{\texttt{fricative}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{fricative}(x)$	$\phi^2_{\texttt{fricative}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{fricative}(x)$
$\phi^1_{\texttt{nasal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{nasal}(x)$	$\phi^2_{\texttt{nasal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{nasal}(x)$
$\phi^1_{\texttt{lateral}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{lateral}(x)$	$\phi^2_{\texttt{lateral}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{lateral}(x)$
$\phi^1_{\texttt{rhotic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{rhotic}(x)$	$\phi^2_{\texttt{rhotic}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{rhotic}(x)$
$\phi^1_{\texttt{labial}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{labial}(x)$	$\phi^2_{\texttt{labial}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{labial}(x)$
$\phi^1_{\texttt{coronal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{coronal}(x)$	$\phi^2_{\texttt{coronal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{coronal}(x)$
$\phi^1_{\texttt{dorsal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{dorsal}(x)$	$\phi^2_{\texttt{dorsal}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{dorsal}(x)$
$\phi^1_{\texttt{voiced}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{voiced}(x)$	$\phi^2_{\texttt{voiced}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{voiced}(x)$
$\phi^1_{\texttt{voiceless}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{voiceless}(x)$	$\phi^2_{\texttt{voiceless}}(x)$	$\stackrel{\text{def}}{=}$	$\texttt{voiceless}(x)$

Table 3.4: The complete logical specification of total reduplication when the input and output string models are both the feature-based successor model.

3.3 Power of MSO-definable Transformations

What other kinds of transformations can be described with logical transformations? As the astute reader may no doubt have already gathered, many phonologically or morphologically *unnatural* processes are also easy to describe with logical transformations. As explained more detail in the next chapter, this is a strength, not a weakness, of the formal methods advocated here. Basically, the formal methods do not constitute a *theory* of phonology;

DRAFT

rather, they constitute a *meta-language* in which theories of phonology can be stated.

In this section, however, we simply wish to establish the fact that two unnatural processes – string mirroring and sorting – also permit logical descriptions.

3.3.1 Mirroring

String mirroring is a process that takes any string w as input and outputs ww^r where w^r is the reverse of the string w. For example if the string *pan* is submitted to the mirroring process, then the output would be *pannap*. Similarly, if *paka* were input to the mirroring process, the output would be *pakaakap*.

This can be modeled with a logical transduction that is nearly identical to the one for total reduplication. The unary relations are defined in the same way. The only differences lie in two of the formulas for the successor relation in the output model. Specifically we define $\phi^{1,1}_{\lhd}(x,y)$ and $\phi^{2,1}_{\lhd}(x,y)$ as before. However, $\phi^{2,2}_{\lhd}(x,y) \stackrel{\text{def}}{=} y \lhd x$, which essentially reverses the successor relations in the second copies of the domain elements. Also, $\phi^{1,2}_{\lhd}(x,y) \stackrel{\text{def}}{=} \mathsf{word\text{-}final}(x) \wedge \mathsf{word\text{-}final}(y)$. Thus mirroring places the copies of the element which is word-final in the input model into the successor relation. Figure 3.8 shows the output model of the string *pannap* that is produced by this logical description of string mirroring given the input *pan*.

3.3.2 Sorting

String sorting is a process that takes any string as input and outputs a string of the same length where the symbols are sorted in leixographic order. For instance if the input string is *paka* the output string would be *aakp*. Similarly, if the input string was *banapi* the output string would be *aabinp*. While, we can do this for any word model of strings, we will assume an alphabet Σ and a conventional precedence model for strings (see section 2.7) for convenience. We also assume that the alphabet is totally ordered under some lexicographic order ($<_\ell$).

Then sorting can be modeled with a logical transduction as follows. We let the copyset $C = \Sigma$. This may seem unusual, but what we are saying is that we make as many copies as there are letters in the alphabet. Then for

 © Jeffrey Heinz

DRAFT

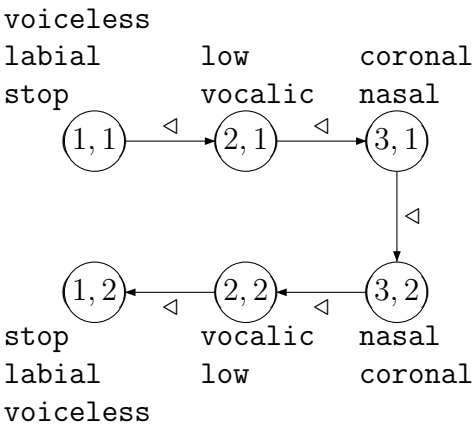

Figure 3.8: The model representing *pannap* is the output of the input *pan* to the process of mirroring.

DRAFT

each $a \neq b \in \Sigma$ define the licensing formulas and the relational formulas as follows.

- $\phi^{b}_{\mathtt{a}}(x) \stackrel{\text{def}}{=} \mathtt{a}(x)$
- $\phi^{a,a}_{<}(x,y) \stackrel{\text{def}}{=} x < y$
- $\phi^{a,b}_{<}(x,y) \stackrel{\text{def}}{=} \mathit{true}$ whenever $a <_\ell b$ and `false` otherwise
- $\phi^{a}_{\mathtt{license}}(x) \stackrel{\text{def}}{=} \mathtt{a}(x)$

The first item defines all the unary relations in the output (for each $a \in \Sigma$). It says that each copy of a domain element in the input which belonged to the unary relation `a` also belongs to `a` in the output.

The second item defines the binary precedence relations for domain elements in the output that belong to the same copy. In this case, domain elements (x, a) and (y, a) stand int the precedence relation in the output only if $x < y$ in the input. This ensures the familiar left-to-right ordering among elements, at least within a copy.

The third item defines the binary precedence relations for domain elements in the output that belong to different copies. The basic idea is that alhabetically earlier copies will precede alphabetically later ones. Recall we are defining multiple formulas $\phi^{a,b}_{<}(x,y)$ for each $a \neq b \in \Sigma$. Whenever $a <_\ell b$

 © Jeffrey Heinz

(a is alphabetically earlier than b) we let $\phi_{<}^{a,b}(x,y) \stackrel{\text{def}}{=} \texttt{true}$. Whenever $b <_\ell a$, we let $\phi_{<}^{a,b}(x,y) \stackrel{\text{def}}{=} \texttt{false}$.

Finally, we get to the licensing formulas, of which there will be $|\Sigma|$. We define these formulas so that only those domain elements that belong to the unary relation a in the ath copy are licensed. Everything else is unlicensed. Recall that relations in the output model are restricted to the licensed domain elements.

Figure 3.9 illustrates this construction when the input is *paka*.

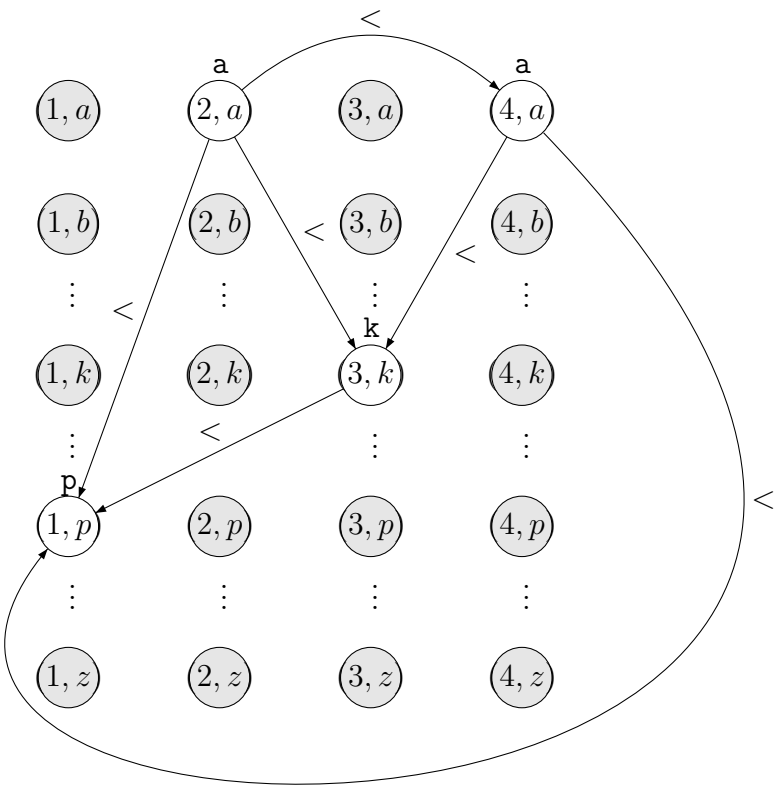

Figure 3.9: The model representing the output *aakp* of the input *paka* to the process of sorting with all potential domain elements shown. Unlicensed elements are in gray.

DRAFT

 © Jeffrey Heinz

3.4 Conclusion

This chapter has explained how transformations can be expressed logically between model-theoretic representations. The signature of the output model determines the relational formulas that need to be defined.

DRAFT

 © Jeffrey Heinz

DRAFT

Part II

Case Studies

DRAFT

DRAFT

© Jeffrey Heinz

DRAFT

Part II

Theoretical Contributions

DRAFT

DRAFT

Part III

Horizons

DRAFT

Bibliography

Albro, Dan. 2005. A large-scale, LPM-OT analysis of Malagasy. Doctoral dissertation, University of California, Los Angeles.

Anderson, Stephen. 1974. *The Organization of Phonology*. Academic Press.

Bale, Alan, and Charles Reiss. 2018. *Phonology: A Formal Introduction*. The MIT Press.

Beesley, Kenneth, and Lauri Kartunnen. 2003. *Finite State Morphology*. CSLI Publications.

Benua, Laura. 1997. Transderivational identity: Phonological relations between words. Doctoral dissertation, University of Massachusetts, Amherst.

Büchi, J. Richard. 1960. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly* 6:66–92.

Chandlee, Jane. 2014. Strictly local phonological processes. Doctoral dissertation, The University of Delaware.

Chomsky, Noam, and Morris Halle. 1968a. *The Sound Pattern of English*. New York: Harper & Row.

Chomsky, Noam, and Morris Halle. 1968b. *The Sound Pattern of English*. New York: Harper & Row.

Courcelle, Bruno. 1994. Monadic second-order definable graph transductions: a survey 126:53–75.

Dresher, Elan B. 2011. The phoneme. In *The Blackwell Companion to Phonology*, edited by Elizabeth Hume Marc van Oostendorp, Colin J. Ewen and Keren Rice, vol. 1, 241–266. Malden, MA & Oxford: Wiley-Blackwell.

DRAFT

Droste, Manfred, and Paul Gastin. 2009. Weighted automata and weighted logics. In Droste *et al.* (2009), chap. 5.

Droste, Manfred, and Werner Kuich. 2009. Semirings and formal power series. In Droste *et al.* (2009), chap. 1.

Droste, Manfred, Werner Kuich, and Heiko Vogler, eds. 2009. *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer.

Frank, Robert, and Giorgo Satta. 1998. Optimality Theory and the generative complexity of constraint violability. *Computational Linguistics* 24:307–315.

Gerdemann, Dale, and Mans Hulden. 2012. Practical finite state optimality theory. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, 10–19. Donostia–San SebastiÃ¡n: Association for Computational Linguistics.
URL `http://www.aclweb.org/anthology/W12-6202`

Goodman, Joshua. 1999. Semiring parsing. *Computational Linguistics* 25:573–606.

Hansson, Gunnar. 2010. *Consonant Harmony: Long-Distance Interaction in Phonology*. No. 145 in University of California Publications in Linguistics. Berkeley, CA: University of California Press. Available on-line (free) at eScholarship.org.

Hayes, Bruce, Bruce Tesar, and Kie Zuraw. 2013. Otsoft 2.3.2. software package.
URL `http://www.linguistics.ucla.edu/people/hayes/otsoft`

Hopcroft, John, Rajeev Motwani, and Jeffrey Ullman. 2001. *Introduction to Automata Theory, Languages, and Computation*. Boston, MA: Addison-Wesley.

Hulden, Mans. 2009a. Finite-state machine construction methods and algorithms for phonology and morphology. Doctoral dissertation, University of Arizona.

DRAFT

Hulden, Mans. 2009b. Foma: a finite-state compiler and library. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, 29–32. Association for Computational Linguistics.

Johnson, C. Douglas. 1972. *Formal Aspects of Phonological Description*. The Hague: Mouton.

Kager, René. 1999. *Optimality Theory*. Cambridge University Press.

Kaplan, Ronald, and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20:331–378.

Karttunen, Lauri. 1998. The proper treatment of optimality in computational phonology. In *FSMNLP'98*, 1–12. International Workshop on Finite-State Methods in Natural Language Processing, Bilkent University, Ankara, Turkey.

Karttunen, Lauri. 2006. The insufficiency of paper-and-pencil linguistics: the case of Finnish prosody. Rutgers Optimality Archive #818-0406.

Kenstowicz, Michael, and Charles Kisseberth. 1977. *Topics in Phonological Theory*. New York: Academic Press.

Kenstowicz, Michael, and Charles Kisseberth. 1979. *Generative Phonology*. Academic Press, Inc.

de Lacy, Paul. 2011. Markedness and faithfulness constraints. In *The Blackwell Companion to Phonology*, edited by M. V. Oostendorp, C. J. Ewen, E. Hume, and K. Rice. Blackwell.

McCarthy, John. 2003. OT constraints are categorical. *Phonology* 20:75–138.

McCarthy, John. 2008. *Doing Optimality Theory*. Malden, MA: Blackwell.

McCarthy, John, and Alan Prince. 1995. Faithfulness and reduplicative identity. In *Papers in Optimality Theory*, edited by Jill Beckman, Laura Walsh Dickey, and Suzanne Urbanczyk, no. 18 in University of Massuchusetts Occasional Papers in Linguistics, 249–384.

McNaughton, Robert, and Seymour Papert. 1971. *Counter-Free Automata*. MIT Press.

DRAFT

Mohri, Mehryar, and Richard Sproat. 1996. An efficient compiler for weighted rewrite rules. In *Proceedings of the 34th Meeting of the Association for Computational Linguistics (ACL '96)*.

Odden, David. 1994. Adjacency parameters in phonology. *Language* 70:289–330.

Odden, David. 2014. *Introducing Phonology*. 2nd ed. Cambridge University Press.

Prince, Alan. 2002. Entailed ranking arguments. In *Rutgers Optimality Archive*. ROA-500, http://roa/rutgers.edu.

Prince, Alan, and Paul Smolensky. 1993. Optimality Theory: Constraint interaction in generative grammar. Tech. Rep. 2, Rutgers University Center for Cognitive Science.

Prince, Alan, and Paul Smolensky. 2004. *Optimality Theory: Constraint Interaction in Generative Grammar*. Blackwell Publishing.

Prince, Alan, Bruce Tesar, and Nazarré Merchant. 2016. Otworkplace. software package. Additions by Luca Iacoponi and Natalie DelBusso. URL `https://sites.google.com/site/otworkplace/home`

Riggle, Jason. 2004. Generation, recognition, and learning in finite state Optimality Theory. Doctoral dissertation, University of California, Los Angeles.

Rogers, James, and Geoffrey Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information* 20:329–342.

Savitch, Walter J. 1993. Why it may pay to assume that languages are infinite. *Annals of Mathematics and Artificial Intelligence* 8:17–25.

Scobbie, James M., John S. Coleman, and Steven Bird. 1996. Key aspects of declarative phonology. In *Current Trends in Phonology: Models and Methods*, edited by Jacques Durand and Bernard Laks, vol. 2, 685–709. Manchester, UK: European Studies Research Institute. University of Salford.

DRAFT

Shukla, Shaligram. 2000. *Hindi Phonology*. Muenchen: Lincom Europa.

Sipser, Michael. 1997. *Introduction to the Theory of Computation*. PWS Publishing Company.

Staubs, Robert, Michael Becker, Christopher Potts, Patrick Pratt, John J. McCarthy, and Joe Pater. 2010. Ot-help 2.0. software package. URL `http://people.umass.edu/othelp/`

Tesar, Bruce. 2014. *Output-driven Phonology*. Cambridge University Press.

Thomas, Wolfgang. 1982. Classifying regular events in symbolic logic. *Journal of Computer and Systems Sciences* 25:370–376.

DRAFT

 © *Jeffrey Heinz*